VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

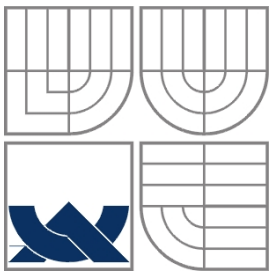
ZJIŠŤOVÁNÍ IZOMORFIZMU GRAFŮ V DATABÁZI

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

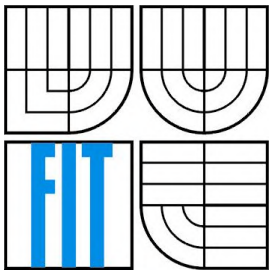
AUTOR PRÁCE
AUTHOR

Bc. ROMAN STEJSKAL

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ZJIŠŤOVÁNÍ IZOMORFIZMU GRAFŮ V DATABÁZI

GRAPH ISOMORPHISM PROBLEM IN DATABASES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. ROMAN STEJSKAL

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PETR CHMELAŘ

BRNO 2008

Abstrakt

Tato práce seznamuje s historií a základními pojmy teorie grafů. Popisuje problémy teorie grafů, možnou reprezentaci grafů a praktickou správu grafů v databázích. Zaměřuje se na problém izomorfizmu podgrafů a grafů. Uvádí možná řešení nalezení izomorfizmu grafů a popisuje vybrané algoritmy pro zjišťování izomorfizmu. V experimentální části se zaměřuje na porovnání dvou implementovaných algoritmů. A to na Ullmannův a VF2 algoritmus. Dále zkoumá rozdíl mezi grafem uloženým v paměti a v databázi.

Klíčová slova

Teorie grafů, reprezentace grafů, problémy teorie grafů, problém izomorfizmu, správa grafů v databázi, izomorfizmus podgrafů a grafů v databázi

Abstract

This project introduces history and basic notions of the graph theory. It describes graph theory problems, possible graph representations and practical graph management in databases. Aims to subgraph and graph isomorphism. It describes possible ways to find graph isomorphism and chosen algorithms for subgraph and graph isomorphism. The experimental part aims to comparing two implemented algorithms. These are Ullmann and VF2 algorithm. Also searches difference between graphs stored in memory and graphs stored in database.

Keywords

Graph theory, graph theory problems, isomorphism problem, management graphs in database, subgraph and graph isomorphism in database

Citace

Stejskal Roman: Zjišťování izomorfizmu grafů v databázi. Brno, 2008, diplomová práce, FIT VUT v Brně.

Zjišťování izomorfizmu grafů v databázi

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Petra Chmelaře. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Roman Stejskal
. května 2008

Poděkování

Chtěl bych poděkovat vedoucímu mé diplomové práce Ing. Petru Chmelařovi za odbornou a technickou pomoc při řešení diplomové práce. Neméně bych rád poděkoval své rodině a přátelům za podporu a pomoc při psaní této práce.

Obsah

1 Úvod

Teorie grafů je v dnešní době známý pojem, přesto že nepatří mezi nejstarší matematické disciplíny. Za svou téměř 300 letou historii udělala velký pokrok a v dnešní době se s ní můžeme setkat v mnoha vědních a technických oborech jako jsou např. bioinformatika, počítačové vidění, počítačové sítě, elektrické obvody, chemie, geografie a mnoho dalších. Podstatou teorie grafů je graf, který je reprezentován objekty a vztahy mezi nimi. Příkladem objektu může být město a vztahem mezi městy silnice. Tento příklad se dá aplikovat na spoustu věcí z reálného života, které se dále snažíme nějak popsat ve virtuálním světě. Proto je dobré se této problematice věnovat. Teorie grafů nám může v mnohém ulehčit práci.

Teorie grafů se zabývá několika problémy, jako je např. problém obarvení, izomorfizmu, obchodního cestujícího, hledání minimální kostry a podobně. Tyto problémy nám můžou pomoci například při hledání nejkratší cesty z jednoho města do druhého. Pro tuto práci je důležitý problém izomorfizmu grafu, jenž se zabývá zjištěním, zda-li jsou dva grafy stejné až na jejich označení. Pro experimenty jsem vybral dva algoritmy. A to Ullmannův algoritmus a algoritmus VF2 navrhnutý pro hledání izomorfizmu rozsáhlých grafů. Pro reprezentaci grafu, na kterém jsem prováděl experimenty byla zvolena česká Wikipedie, přesněji její graf odkazů.

V první kapitole seznamuji s historií vývoje teorie grafů. Následující část si klade za úkol osvětlit základní pojmy důležité k pochopení další části zabývající se zjišťováním izomorfizmu grafů. Další kapitola pojednává o praktické správě grafů v databázích. Poté je snaha o identifikaci problému izomorfizmu, diskuze o možných řešeních tohoto problému a představení vybraných algoritmů pro nalezení izomorfizmu. Následuje kapitola, která popíše implementaci vybraných algoritmů pro hledání izomorfizmu podgrafů. Poté jsou představeny a rozebrány experimenty s implementovanými algoritmy.

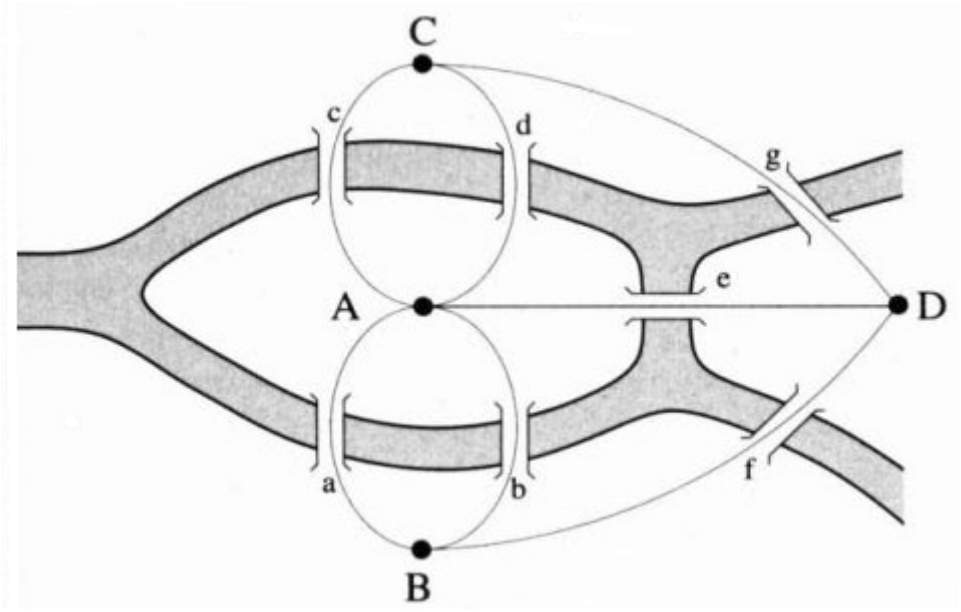
Tato práce navazuje na semestrální projekt, který se zabýval převážně teoretickou částí zadání diplomové práce, do níž spadá teorie grafů, správa grafů v databázi, izomorfizmus a výběr vhodných algoritmů pro řešení diplomové práce.

2 Úvod do teorie grafů

Oblast teorie grafů je v dnešní době rozšířená a probádaná. Proto je vhodné se na začátek seznámit s jejím historickým vývojem a dále osvětlit některá její zákoutí, která jsou důležitá pro její pochopení a také pro lepší porozumění tématu, jímž se tato práce zabývá. Dále tedy budou popsány základní pojmy z oblasti teorie grafů. Druhá kapitola seznamuje čtenáře s grafem, jeho definicí, vlastnostmi a několika jeho typy. V navazující kapitole se zaměří na reprezentaci grafů. Další teoretická část se již bude zabývat hlavním problémem této práce, a to izomorfizmem grafů.

2.1 Stručná historie teorie grafů

Teorie grafů není zase tak stará matematická disciplína oproti např. geometrii, algebře, analýze apod. Za jejího zakladatele je považován **Leonhard Euler** (1707-1783), který se v roce 1736 ve své práci *Solutio problemas ad geometriam situs pertinentis* zabýval řešením úlohy sedmi mostů Königsbergu. Podstatou úlohy bylo projít cestu po Königsbergu (viz Obrázek 2.1) tak, aby po každém z jeho mostů přešel právě jednou a vrátil se na místo odkud vyšel. Pro tuto úlohu vytvořil podmínky a topologii nutné k absolvování této cesty.



Obrázek 2.1: Sedm mostů Königsbergu (převzato z: [Obr1])

Po více jak 100 letech se grafy začal zabývat **Gustav Kirchhoff** (1824-1887), jenž v roce 1845 publikoval zákony pro elektrické obvody. Pomocí těchto zákonů se dá vypočítat proud a napětí v jednotlivých větvích elektrického obvodu. Další osobou kdo se zabývala grafy byl anglický matematik **Arthur Cayley** (1821-1895), který řešil problém kolik existuje izomerů uhlovodíku

C_nH_{2n+2}. Použil grafickou abstrakci, aby znázornil atomy a vazby mezi nimi. Podle něj začala chemie používat grafické znázornění chemických vzorců.

Obrázek 2.2: Problém čtyř barev (převzato a upraveno z: [Obr2])

Roku 1852 předložil **Francis Guthrie** tzv. problém čtyř barev, při pokusu vytvořit mapu anglických hrabství. Na obrázku 2.2 je vidět možný příklad takové mapy. Vyslovil otázku, je-li možné vytvořit mapu obsahující plošné elementy, které budou vybarveny čtyřmi barvami právě tak, aby dvě sousední plochy neměly nikdy stejnou barvu. Tento problém se podařilo vyřešit až v roce 1977 **Kennethem Appelem** a **Wolfgangem Hakenem** za pomoci moderní výpočetní techniky. Pro vyřešení problému bylo zavedeno spoustu důležitých konceptů teorie grafů. Např. rovinný graf.

V roce 1859 sir **William Rowan Hamilton** (1805-1865) vymyslel hru jmenující se „Icosian Game“. Jednalo se o hledání hamiltonovských kružnic v grafu, který se rovná pravidelnému dvanáctistěnu.

V roce 1878 **James Joseph Sylvester** (1814-1897) jako první použil pojem graf v tom smyslu, jak mu rozumíme v dnešní době. V 3. díle *Encyklopädie der mathematischen Wissenschaften* nalezneme první přehled nejstarších výsledků teorie grafů. **Max Wilhelm Dehn** (1878-1952) a **Poul Heegaard** (1871-1948) zde v části pojmenované Analysis Situs zasvěcují čtenáře do problematiky cest v grafech, vlastností stromů, rozkladu grafů, problému čtyř barev a několika dalších problémů týkajících se grafů. První knihou zabývající se pouze teorii grafů napsal maďarský matematik **Dénese König** (1884-1944). Vyšla v roce 1936 v Lipsku pod názvem *Theorie der endlichen und unendlichen Graphen* a König v ní systematizoval teorii grafů. Po dlouhá léta pak tato kniha sloužila jako jediná učebnice teorie grafů.

Až někdy na konci 50. a 60. let se objevují další knihy zabývající se teorii grafů. V těchto letech se teorie grafů rychle rozvíjí. A její vývoj trvá do dnešních dob. Mezi nejznámější knihy patří

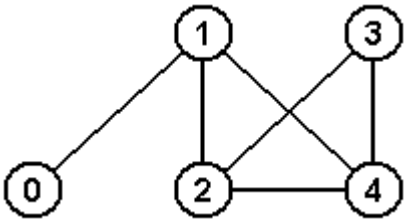
Théorie des Graphes et ses Applications, která vyšla v roce 1958 a jejím autorem je francouzský matematik **Claud Berge**. Další kniha vychází v roce 1962 pod názvem *Theory of Graphs*. Autorem je norský matematik **Oystein Oreh** (1899-1968). Roku 1969 vydává americký matematik **Frank Harary** knihu věnovanou teorii grafů pod jménem *Graph Theory*. Následuje období, ve kterém přichází řada publikací o teorii grafů a taky o speciálních částech teorie grafů.

2.2 Graf

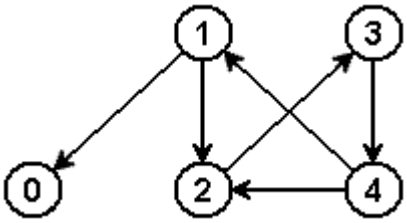
Pro následující teoretickou část zabývající se základními definicemi a vlastnostmi grafu jsem čerpal z následujících materiálů [Ch06], [BM82], [Ko84], [Hl07], [Ve07], [Wi08].

Graf v matematice je nejčastěji grafické znázornění funkční závislosti. Ale v teorii grafů je graf chápán jako objekty a vztahy mezi těmito objekty. Objekty jsou zde reprezentovány vrcholy a vztahy se vyjadřují pomocí hran, které spojují vrcholy. Tyto hrany můžou vyjadřovat jednostranné i oboustranné vztahy. Na základě těchto vztahů a jejich kombinací lze rozlišit několik základních typů grafu. A to obyčejný graf, orientovaný graf a obecný graf (multigraf). Další možné typy jsou od těchto grafů odvozeny.
Hrany v grafech mohou být dvojího druhu:

- **Neorientované** – vztahy mezi vrcholy jsou symetrické. Vyjadřují oboustranné vztahy. (viz Obrázek 2.3)
- **Orientované** – vztahy mezi vrcholy jsou nesymetrické. Vyjadřují jednostranné vztahy. Pomocí šipky se označuje směr orientace. (viz Obrázek 2.4)



Obrázek 2.3: Neorientovaný graf



Obrázek 2.4: Orientovaný graf

Podle typu hran lze grafy rozdělit na:

- **Neorientované** – mají všechny hrany neorientované
- **Orientované** – mají všechny hrany orientované
- **Smíšené** – obsahují neorientované i orientované hrany

Obyčejný graf *G* je definován jako uspořádaná dvojice *(V, E)*, kde *V* je neprázdná množina vrcholů grafu *G* a *E* se nazývá množina hran grafu *G* a platí :

$$E \subseteq \binom{V}{2} \tag{2.1}$$

Orientovaný graf *G* je definován jako obyčejný graf jehož všechny hrany jsou orientovány.

Obecný graf *G* je definován jako trojice *(V, E, ε)*, kde *V* je konečná množina vrcholů, *E* je konečná množina hran a *ε* je zobrazení, které přiřazuje každé hraně dvojici různých uzlů, tedy:

$$\varepsilon : E \to \{(u, v) : u, v \in V\} \tag{2.2}$$

Pokud mezi jednou dvojicí uzlů může být několik hran nebo hrana začíná a končí ve stejném vrcholu, jedná se o tzv. **multigraf**.

Existují i další typy grafů, které se definují podobně. Například **smíšený graf** je graf, který obsahuje jak orientované hrany, tak i neorientované hrany. Další možné grafy jsou jejich různé kombinace.

U grafů se můžeme setkat s dalšími pojmy, které se v následující části pokusím objasnit:

- **Sled** – je v grafu *G = (V, E)* posloupnost vrcholů v_i a hran e_i $P = (v_1, e_1, v_2, e_2, ..., e_{n-1}, v_n)$, kde $v_1, v_2, ..., v_n \in V$ a $e_1, e_2, ..., e_n \in E$ a pro niž platí $e_i = \{v_i, v_{i+1}\}$ nebo $e_i = (v_i, v_{i+1})$ pro orientovaný graf. Jinými slovy to je posloupnost vrcholů, které jsou spojené hranami způsobem takovým, že každý vrchol je spojen hranou právě s tím vrcholem, jenž tento vrchol následuje v dané posloupnosti.
- **Tah** – je sled mezi vrcholy v_1 a v_n o délce *n*, kde se mohou opakovat uzly, ale všechny hrany jsou různé.
- **Cesta** – je sled mezi vrcholy v_1 a v_n o délce *n*, kde se neopakují vrcholy ani hrany.
- **Kružnice** – je uzavřená cesta pro níž platí $v_1 = v_n$.
- **Vzdálenost uzlů v_1 a v_n** – je počet hran nejkratší cesty z vrcholu v_1 do v_n.
- **Souvislý graf** – je takový graf, pro který platí, že mezi jeho dvěma libovolnými uzly existuje sled a tedy i cesta. Rozeznáváme slabou a silnou souvislost. O slabou souvislost se jedná při symetrizaci(deorientaci) orientovaného grafu. Orientovaný graf je silně souvislý tehdy, kdy pro libovolné dva uzly $u, v \in V$ existuje orientovaná cesta z uzlu *u* do uzlu *v* a rovněž opačná cesta z uzlu *v* do uzlu *u*.
- **Komponenta** – je v nesouvislém grafu každý maximální souvislý podgraf.

- **Úplný graf** – je takový, v němž jsou každé dva vrcholy spojené hranou. Označuje se K_n, kde *n* je počet vrcholů a počet hran je $\binom{n}{2}$.

- **Bipartitní graf** – je graf, jehož množinu vrcholů je možné rozdělit na dvě disjunktní množiny tak, že vrcholy ze stejné množiny nejsou spojeny hranami.

- **Podgraf** – podgrafem grafu G je libovolný graf H na podmnožině vrcholů $V(H) \subseteq V(G)$, který má za hrany libovolnou podmnožinu hran grafu G mající oba vrcholy ve $V(H)$. Zapisujeme $H \subseteq G$, tj. stejně jako množinová inkluze, ale význam je trochu odlišný. Viz [Hl07]. Nebo jinými slovy to je graf, který je izomorfní (viz kapitola 2.4) s nějakou částí nadgrafu.

- **Stupeň vrcholu** – je-li graf $G=(V, E)$ obyčejný graf a $u \in V$, potom je stupněm vrcholu nezáporné číslo označováno $deg(u)$ definováno jako počet hran incidentních s uzlem u. Pokud platí $deg(u)=0$, je vrchol $u \in V$ nazýván izolovaným. U orientovaných grafů rozeznáváme vstupní a výstupní stupeň uzlu. Vstupní stupeň uzlu $deg_+(u)$ se rovná počtu hran, které vedou z nějakého uzlu do uzlu u. Výstupní stupeň uzlu $deg_-(u)$ se rovná počtu hran, které vedou z uzlu u do nějakého uzlu.

- **Skóre grafu** – je nerostoucí posloupnost stupňů vrcholů obyčejného grafu G:

$$D = (\deg(v), v \in V) \tag{2.3}$$

2.3 Invarianty grafu

Graf má určité vlastnosti, které jsou určitým způsobem charakterizovány čísly. Říká se jim číselné invarianty grafu nebo také číselné charakteristiky grafu. Těchto invariantů je v teorii grafů poměrně mnoho.

V [Ch06] je uvedeno, že pokud máme grafy $G=(V, E)$ a $G'=(V', E')$ a chceme-li určit, zda jsou izomorfní, je požadována funkce, která má následující vlastnosti:

(i) pokud platí vztah $\gamma(G) = \gamma(G')$, potom G je izomorfní s G' (2.4)

(ii) pokud je G izomorfní s G', potom platí i $\gamma(G) = \gamma(G')$ (2.5)

Výsledkem této funkce γ je takzvaný **invariant grafu**, jenž může být reprezentován číslem, polynomem nebo maticí.

Invarianty, které splňují podmínku (2.4) uvedené v [Ch06]:

- **Počet hran grafu** – G: $n = |V|$.

- **Počet vrcholů grafu** – $m = |E|$.
- **Nejvyšší a nejnižší stupeň grafu** – *G:* $\Delta(G)$ *a* $\delta(G)$.
- **Průměrný stupeň** – $d(G)$, platí: $\Delta(G) \geq d(G) \geq \delta(G)$.
- **Skóre grafu** – je souhrnný invariant, z něhož je možné vyvodit výše uvedené.

Dalšími zajímavými invarianty jsou matice **adjacence**, **incidence** a **dosažitelnosti**. O nich pojednávají následující kapitoly.

2.3.1 Matice adjacence

Je čtvercová matice konečného neorientovaného nebo orientovaného grafu $G=(V, E)$ velikosti nxn, kde $n=|V|$ a pro $A(G)=[a_{ij}]$ platí:

$$a_{ij} = \begin{cases} 1, pokud\ \{v_i, v_j\} \in E, respektive\ (v_i, v_j) \in E \\ 0\ jinak \end{cases} \tag{2.6}$$

Diagonála a_{ii} je reprezentována smyčkami a v případě, že se jedná o prostý graf je nulová. V případě neorientovaného grafu je matice adjacence symetrická. Na obrázku 2.5 je adjacenční matice grafu z obrázku 2.3 a na obrázku 2.6 je adjacenční matice grafu z obrázku 2.4.

	0	1	2	3	4
0	0	1	0	0	0
1	1	0	1	0	1
2	0	1	0	1	1
3	0	0	1	0	1
4	0	1	1	1	0

Obrázek 2.5: Adjacenční matice pro neorientovaný graf

	0	1	2	3	4
0	0	0	0	0	0
1	1	0	1	0	0
2	0	0	0	1	0
3	0	0	0	0	1
4	0	1	1	0	0

Obrázek 2.6: Adjacenční matice pro orientovaný graf

Matice splňuje podmínku (2.5) pro určení izomorfizmu, ale ne podmínku (2.4). Ale v [Ch06] je uvedeno, že pokud máme adjacenční matice *A(G)* a *A(G')*, pak grafy *G* a *G'* jsou izomorfní pokud existuje permutační matice *P* a platí:

$$PA(G)P^{-1} = A(G') \tag{2.7}$$

A z toho plyne, že *A(G)* a *A(G')* jsou stejné až na permutaci jejich řádků a sloupců. Což je nutná a postačující podmínka pro izomorfizmus grafů (2.4), (2.5) a taky způsob, jak izomorfizmus zjistit. To znamená nalézt permutační matici.

Dále je možné z matice adjacence zjistit spektrum grafu, což je kompletní množina vlastních hodnot, které splňují podmínku (2.4). Příkladem jsou stejné minimální a charakteristické polynomy, determinant a stopa.

2.3.2 Matice incidence

Je matice typu *n*x*m* grafu *G=(V, E)*, kde *n*=|*V*| a *m*=|*E*| a pro *B(G)=[b_{ij}]* platí:

$$b_{ij} = \begin{matrix} 1, pokud\ \{v_i, v_j\} \in E \\ 0\ jinak \end{matrix} \tag{2.8}$$

Matice incidence má podobné vlastnosti jako matice adjacence. V [Ch06] je uveden zajímavý vztah pro hodnost této matice:

$$H(B(G)) = |U| - p \tag{2.9}$$

kde *p* je počtem komponent. Matice incidence ale neumožňuje tyto komponenty určit a pro určení izomorfizmu je třeba *n!m!* permutací. Kdežto u adjacenční matice je těchto permutací *n!*. Z toho důvodu se matice incidence pro izomorfizmus nevyužívá. Dále se pomocí ní špatně zaznamenávají smyčky a tudíž se nedá pracovat s orientovanými grafy a multigrafy.

	0-1	1-2	2-3	3-4	1-4	2-4
0	1	0	0	0	0	0
1	1	1	0	0	1	0
2	0	1	1	0	0	1
3	0	0	1	1	0	0
4	0	0	0	1	1	1

Obrázek 2.7: Matice incidence

Na obrázku 2.7 je znázorněna matice incidence pro graf z obrázku 2.3.

2.3.3 Matice dosažitelnosti

Je matice typu *n*x*n* grafu *G=(V, E)*, kde *n=|V|* a pro $R(G)=[r_{ij}]$ platí:

$$r_{ij} = \begin{cases} 1, \textit{pokud existuje cesta z } v_i \textit{ do } v_j \\ 0 \textit{ jinak} \end{cases} \tag{2.10}$$

její výpočet je uveden v [Ko84]. V [Ch06] je řečeno, že matice dosažitelnosti umožňuje přímočarým způsobem určit spojité komponenty grafu, oddělit je a zjišťování izomorfizmu aplikovat na tyto bloky.

2.3.4 Matice vzdálenosti

Má podobné vlastnosti jako matice adjacence. Značí se $D(G)=[d_{ij}]$:

$$d_{ij} = \textit{ vzdálenost z } v_i \textit{ do } v_j \tag{2.11}$$

Sestrojení se provádí pomocí Dijkstrova [Ko84], nebo taky pomocí Floyd-Warshallova algoritmu [Wi08].

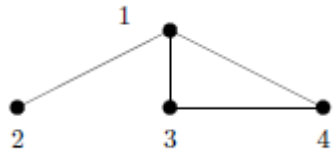
2.3.5 Další invarianty

Invarianty uvedené v předchozím textu nejsou zdaleka všechny možné invarianty, které existují. Např. na následující stránce: [Inv] je projekt, kterým je sbírka kódů pro zjištění různých invariantů grafu. Těchto invariantů je zde momentálně kolem 50. Pro představu dalších možných invariantů uvedu ještě několik dalších.

- **Uzlové chromatické číslo** – je minimální počet barev nutný k obarvení všech uzlů.
- **Hranové chromatické číslo** – je minimální počet barev nutný k obarvení hran.
- **Pokrytí uzlů** – je takový nejmenší počet vrcholů, že každá hrana inciduje alespoň s jedním vrcholem.
- **Pokrytí vrcholů** – je takový minimální počet hran, aby sousedily se všemi vrcholy.
- **Maximální klikovost** – je největší indukovaný úplný graf.
- **Počet klik v grafu** – je počet úplných podgrafů v grafu.
- **Počet trojúhelníků v grafu** – je počet klik o velikosti 3 v grafu.

2.4 Reprezentace grafů

Pojmenování graf je odvozeno z grafické povahy jeho zobrazení. Vrcholy jsou reprezentovány body v prostoru a hrany jsou reprezentovány čárami, které spojují některé z vrcholů. Čáry mohou být jak rovné, tak i zakřivené. Jednoznačný způsob pro kreslení grafů však není dán. Příklad zobrazení grafu v dvourozměrném prostoru je znázorněn na Obrázku 2.4.



Obrázek 2.8: Reprezentace grafu (převzato z [Hl07])

Několik možných a hojně využívaných způsobů reprezentace grafu uvádím v následujících dvou kapitolách, kde se v první kapitole jedná o reprezentaci užívanou v matematice a ve druhé kapitole popíši jak lze grafy popsat v programech.

2.4.1 Reprezentace grafů v matematice

V matematice se můžeme setkat s následujícími nejpoužívanějšími možnostmi, jak popsat graf:

- **Definicí** – graf lze popsat i určitým matematickým vyjádřením. Nejenom grafickým znázorněním. Jedna z možných definic je výčet vrcholů a hran. Příkladem takového výčtu může být následující popis. $V = \{1,2,3,4\}$, $E = \{\{1,2\},\{1,3\},\{3,4\},\{1,4\}\}$. Tento popis reprezentuje graf zobrazený na obrázku 2.8. V je zde výčtem vrcholů a E je výčtem hran, tj. dvojicí vrcholů, které jsou spojeny hranou.

- **Maticemi** – reprezentace grafu pomocí matic vychází ze dvou základních vztahů v grafu. Prvním vztahem je incidence, což je vztah mezi hranou a jejím koncovým uzlem. Ten popisuje matice incidence grafu. Druhý vztah je sousednost uzlů, který popisuje matice adjacence grafu [Ve07]. Podrobnější popis těchto matic je uveden v kapitole o Invariantech grafu.

- **Diagramem** – tento způsob byl již naznačen dříve. Jedná se o grafické znázornění bodů a křivek v prostoru, které tyto body spojují. Přitom nezáleží na způsobu kreslení bodů ani čar. Bod můžeme znázornit jako vyplněný nebo prázdný kruh. Čáry mohou být rovné nebo různými způsoby lomené nebo zakřivené. Pokud se jedná o graf, který má jednostranné vztahy mezi objekty, tak se tyto vztahy zobrazují pomocí šipky na konci čáry. To znamená objekt z něhož šipka vychází má nějaký vztah k objektu, do kterého šipka míří.

2.4.2 Reprezentace grafů v programech

Aby jsme mohli graf použít v našem programu, je třeba tento graf nějakým způsobem uložit. Některé způsoby vycházejí z matematické reprezentace grafů. Ale při programování je třeba brát ohled i na to, kolik paměti daná reprezentace zabere v počítači nebo kolik času je třeba k získání potřebných dat z grafu. Také se bere v potaz algoritmus, který máme v plánu použít. Proto je dobré zvolit vhodnou datovou strukturu (pole, dynamické seznamy, apod.) s ohledem na to, co je pro nás důležité. Zda čas přístupu nebo paměťová náročnost nebo obojí, po případě vhodnost pro zvolený algoritmus. Nyní několik možných způsobů popíši a teoreticky rozeberu. Praktické ověření je součástí jednoho z provedených experimentů, které budou zmíněny později.

Nejčastěji se používají dva obvyklé způsoby uložení grafů v počítači. Jedná se o adjacenční matici reprezentovanou pomocí dvourozměrného pole a adjacenční seznam reprezentován pomocí dynamických seznamů.

Adjacenční matice – je v programech vytvořena pomocí dvourozměrného pole n x n, kde n je počtem vrcholů příslučného grafu. Toto pole je tvořeno jedničkami a nulami. V případě multigrafu je pole tvořeno nulami a kladnými čísly. Její podrobnější popis a sestrojení uvádím v kapitole o invariantech grafu. Z hlediska programování má tato reprezentace prostorovou složitost $O(n^2)$ a konstantní časovou složitost pro vyhledávání. Příklad matice je na obrázcích 2.5 a 2.6.

Adjacenční seznam – nejčastěji se vytváří pomocí dynamického seznamu vrcholů, kde každá položka (vrchol) obsahuje další dynamický seznam obsahující sousedící vrcholy daného vrcholu. Jak takovýto seznam seznamů vypadá můžeme vidět na obrázcích 2.9 a 2.10. Adjacenční seznam má prostorovou složitost $O(m)$, kde m je počtem hran grafu. Časová složitost vyhledávání v grafu je v nejhorším případě $O(n)$.

0	2		
1	1	3	5
2	2	4	5
3	3	5	
4	2	3	4

Obrázek 2.9: Adjacenční seznam pro norientovaný graf

0		
1	1	3
2	4	
3	5	
4	2	3

Obrázek 2.10: Adjacenční seznam pro orientovaný graf

Z paměťového hlediska je tedy lepší adjacenční seznam. Adjacenční matice je výhodnější u grafů s co nejvíce propojeními vrcholů, protože u řídkých grafů vytváří hodně redundantních dat. Z časového hlediska je na tom zase lépe adjacenční matice. Tyto teoretické předpoklady jsou v pozdější části této práci podrobeny testům v praxi, jenž tato tvrzení podpoří.

Dále je možné graf ukládat do souboru nebo databáze. V pozdější kapitole popíši jakým způsobem se grafy ukládají a spravují v databázi.

2.5 Problémy v teorii grafů

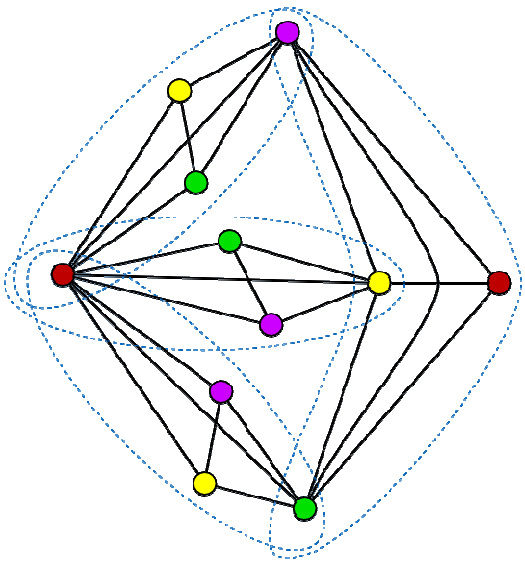
V této kapitole pojednávající o problémech v teorii grafů čerpám z následujících zdrojů [Ko84], [Hl07], [Wi08]. Problémů v teorii grafů je poměrně mnoho a nyní uvedu a popíši některé nejznámější problémy. Tato práce klade hlavní důraz na problém izomorfizmu grafu, proto je pro něj vyhrazena celá 3. kapitola.

2.5.1 Barvení grafu

V teorii grafů se jedná o speciální případ pojmenování grafů (přiřazení jména danému vrcholu nebo hraně). Místo jména se však přiřazuje barva, která se přiřazuje podle daných podmínek. Jednoduše řečeno se jedná o přiřazení barvy vrcholu takovým způsobem, že žádné dva sousedící vrcholy nemají stejnou barvu. Toto se nazývá barvení vrcholů. Obdobně je definováno barvení hran. Plošné barvení rovinného grafu přiřazuje barvu každé ploše tak, že žádné dvě plochy sdílející stejné hranice nemají stejnou barvu. Při barvení grafu G je definováno tzv. **chromatické číslo** grafu, které udává nejmenší možný počet barev, jenž je třeba k obarvení příslušného grafu G.

Spousta problémů v teorii grafů spadá pod barvení grafu. Příkladem těchto problémů jsou následující:

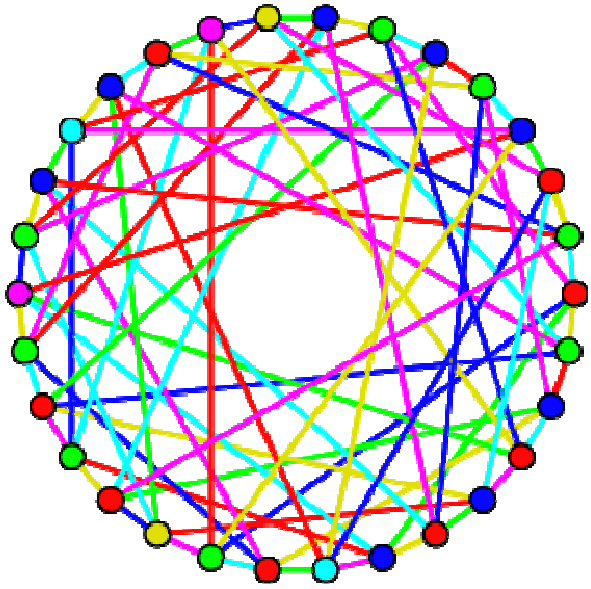
Problém čtyř barev – cílem tohoto problému je obarvit vrcholy planárního grafu tak, aby žádné dva sousedící vrcholy neměly stejnou barvu. Předpokladem je, že pro každý rovinný graf stačí pro obarvení vrcholů jenom čtyři barvy. Tento problém se podařilo Kennethu Appelovi a Wolfgangu Hakenovi vyřešit v roce 1977.



Obrázek 2.11: Erdős-Faber-Lovászova hypotéza (převzato z [Wi08])

Erdős-Faber-Lovászova hypotéza – říká, že sjednocení k kopií klik o velikosti k, které protínají nanejvýše jeden pár vrcholů, je k chromatické. Viz obrázek 2.11. Tento předpoklad zatím nebyl dokázán.

Úplné obarvení – je typem obarvování, při kterém nesmí mít stejnou barvu, žádné dva sousedící vrcholy, žádné incidující hrany a žádný vrchol nesmí mít stejnou barvu jako hrana do něj vcházející nebo z něj vycházející. Příklad takového obarvení je na obrázku 2.12. **Úplné chromatické číslo** grafu G je číslo udávající nejmenší počet barev potřebný pro úplné obarvení grafu G. Tento problém zatím nebyl vyřešen.



Obrázek 2.12: Úplné obarvení (převzato z [Wi08])

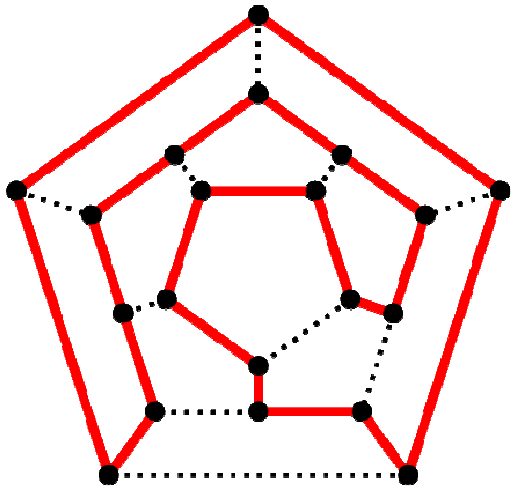
2.5.2 Cestovní problémy

Do této části spadá poměrně hodně problémů. V podstatě se zde jedná o nalezní nějaké cesty v grafu podle stanovených podmínek. Dále uvádím několik představitelů spadajících do této kategorie problémů:

Nejkratší cesta grafu – je takový problém, ve kterém se hledá cesta mezi dvěma vrcholy, kde součet ohodnocení hran v cestě je nejmenší možné. Nejdůležitějšími algoritmy pro nalezení nejkratší cesty jsou Dijkstrův algoritmus, Floyd-Warshallův algoritmus, A* vyhledávací algoritmus, Bellman-Fordův algoritmus a Johnsonův algoritmus. Jejich popis lze nalézt v [Wi08], [OD03]. Tento problém je NP-úplný.

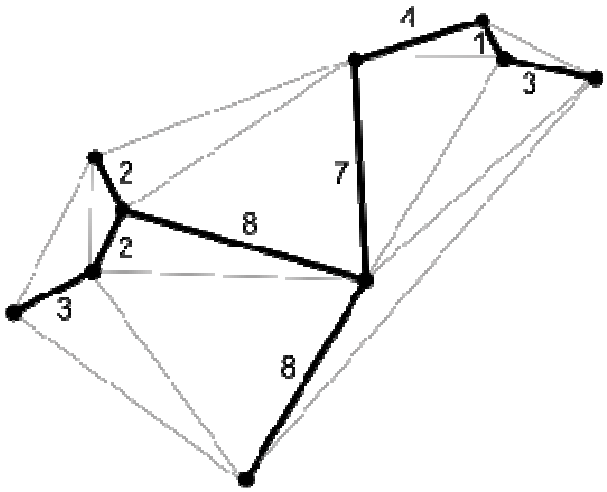
Hamiltonovská kružnice – je to taková cesta, která projde právě jednou všechny vrcholy grafu a zároveň mezi jejíž počátečním a konečným vrcholem existuje platná cesta. Příklad takovéto kružnice

je na obrázku 2.13. Graf se nazývá Hamiltonovský tehdy, obsahuje-li hamiltonovskou kružnici. Problém nalezení této kružnice je NP-úplný.



Obrázek 2.13: Hamiltonovská kružnice (převzato z [Wi08])

Problém obchodního cestujícího – tato úloha souvisí s hamiltonovskou kružnicí. Cílem je nalézt nejkratší trasu pro obchodního cestujícího, který vyjíždí z daného počátečního místa, přitom musí navštívit obchodní místa (alespoň jednou) a nakonec se vrátit na místo odkud vyjel. Úloha je zadána úplným hranově ohodnoceným grafem. Cílem je tedy nalézt nejmenší ze všech hamiltonovských kružnic. Tato úloha patří mezi NP-úplné problémy a proto se pro její řešení používají především heuristické algoritmy poskytující přibližné řešení.



Obrázek 2.14: Minimální kostra grafu (převzato z [Wi08])

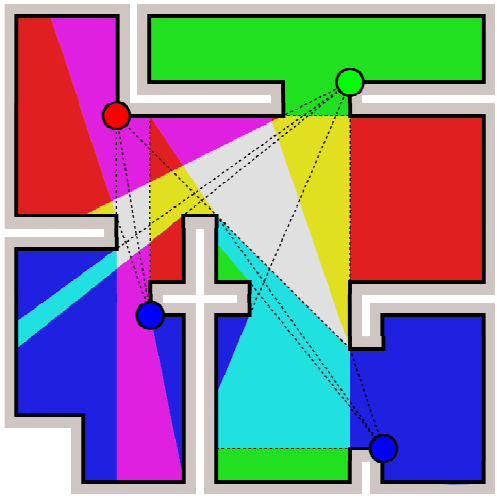
Minimální kostra grafu – jde o nalezení minimální kostry ohodnoceného grafu. Příklad minimální kostry je na obrázku 2.14. Nejznámějšími algoritmy jsou Primův a Kruskalův algoritmus. Jejich popis lze nalézt například v [Wi08].

Problém čínského pošťáka – cílem je nalézt nejkratší trasu pro pošťáka při doručování pošty v jeho obvodě. Je tedy třeba najít uzavřený sled, který pokrývá celý graf a je z takových sledů nejkratší.

Úloha sedmi mostů Königsbergu – úloha je představena v historickém úvodu této práce. Podstatou je rozhodnout, je-li možné projít cestou, která navštíví každý ze sedmi mostů právě jednou a vrátí se zase zpět na počáteční místo. Euler dokázal, že toto není možné.

2.5.3 Problém viditelnosti

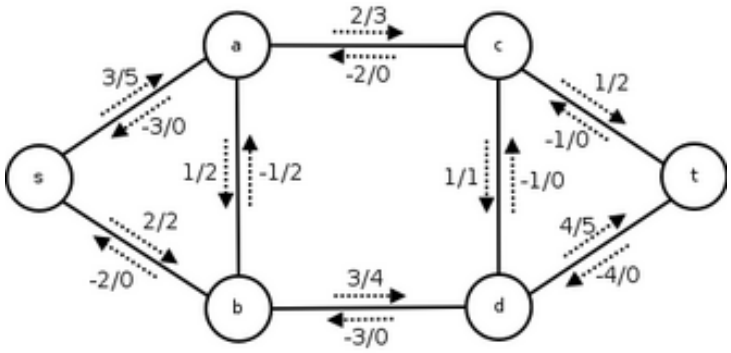
Tento problém se zabývá určením, zda-li je z příslušného místa vidět na určené místo. Představitelem tohoto problému je tzv. problém **galerie umění**. Je to problém ze skutečného světa. Jde o určení minimálního počtu kamer potřebných ke střežení galerie. Obrázek 2.15 zobrazuje problém viditelnosti.



Obrázek 2.15: Problém viditelnosti (převzato z [Wi08])

2.5.4 Toky v sítích

Odvětví zabývající se toky v sítích patří mezi nejbohatší a nejvíce rozpracované aplikace teorie grafů. Reprezentují spoustu problémů, jež se často objevují v informatice, komunikacích a jiných oborech. Model sítě se zde reprezentuje orientovaným grafem, kde se v této síti přepravuje určitá entita po cestách dané sítě. Každá hrana zde má určitou kapacitu a tok, který přijímá. Množství toku nesmí přesáhnout kapacitu dané hrany. Tok v každé hraně je ustálený a beze ztrát. Směr proudění toku je dán orientací hrany. Pokud je velikost toku záporná, jedná se o proudění proti směru hrany. Příkladem takové sítě může být vodovodní, dopravní, spojová síť. Na obrázku 2.16 je příklad toku v sítích. Pomocí jednoduchých algoritmů lze stejným způsobem řešit úlohy z různých odvětví. Například propustnost, minimální tok, nejlevnější tok v počítačové, dopravní nebo kanalizační síti.



Obrázek 2.16: Toky v sítích (převzato z [Wi08])

Toky lze rozlišit na cirkulaci a tok od zdroje ke spotřebiči. Cirkulace splňuje Kirchhoffův zákon pro všechny vrcholy. U toku od zdroje ke spotřebiči platí Kirchhoffův zákon pro všechny vrcholy kromě dvou, kterým říkáme zdroj a spotřebič. Ve zdroji vzniká tok a ve spotřebiči zase zaniká. Více podrobností o toku v sítích přesahuje rámec této práce. Více v [Hl07] a [OD03].

2.6 Prohledávání grafů

Prohledáváním grafů se rozumí průchod všemi uzly grafu. Algoritmy na prohledávání grafů se hojně využívají v různých algoritmech, které řeší nějaký problém z oblasti teorie grafů. Jsou dva základní způsoby jak procházet grafem. Jedním se grafem prochází do hloubky a tím druhým do šířky. V následujících kapitolách tyto způsoby prohledávání grafu popisuji blíže. V této kapitole jsem čerpal informace z [OD03], [Ve07].

2.6.1 Prohledávání grafu do hloubky

Prohledávání grafu do hloubky je realizováno následujícím postupem. Při průchodu grafem je nutné zajistit, abychom navštívili všechny uzly a taky abychom neprocházeli uzlem vícekrát. Uzlům se tedy přiřazuje proměnná, která určuje zda-li jsme již uzel navštívili a provedli v něm potřebné operace. Průchod začínáme libovolným uzlem a přecházíme do kteréhokoliv sousedního uzlu. Tento uzel označíme jako navštívený a zjistíme, jestli lze pokračovat v prohledávání grafu. Pokud ano, přecházíme do dalšího uzlu, který si opět označíme. Nejsou-li v okolí vrcholu dosud neoznačené uzly, vrátíme se nazpátek a zjišťujeme okolí dřívějších uzlů. Tento algoritmus se ukončuje v následujících situacích. Buďto pokud projdeme všemi uzlu nebo pokud dojdeme do předem vybraného vrcholu anebo vrátíme-li se zpět do výchozího uzlu. Při tomto způsobu průchodu grafem se používá zásobník. Více viz [OD03], [Ve07].

2.6.2 Prohledávání grafu do šířky

Při tomto prohledávání grafu algoritmus postupně prochází všechny vrcholy grafu takovým způsobem, že nejprve navštíví všechny sousedy počátečního vrcholu. Poté navštěvuje sousedy sousedů do té doby, než projde celým grafem. Jinými slovy se v každém kroku naleznou všechny uzly, které jsou od počátečního uzlu vzdáleny stejně. Tento algoritmus využívá frontu pro ukládání sousedů v daném vrcholu. Více viz [OD03], [Ve07].

3 Problém izomorfizmu

3.1 Izomorfizmus

Při zobrazování grafů si můžeme položit otázku: *„Je jeden graf nakreslený různými způsoby stále ten stejný graf nebo ne?“* Pokud budeme přísně trvat na slově „stejný“, potom by bylo odpovědí, že jinak nakreslený graf není „stejný“ jako původní. Přitom hlavním úkolem grafů je modelovat vztahy mezi objekty. Takže tyto vztahy nijak nezávisí na grafickém znázornění grafu. Proto existuje pojem *izomorfizmus grafů*, který řeší podobnost grafů na základě vztahů mezi objekty a neřeší jejich grafické zobrazení.

Formální definice izomorfizmu – *izomorfizmus* grafů G a H je bijektivní zobrazení $f : V(G) \to V(H)$, pro které platí, že každá dvojice vrcholů $u, v \in V(G)$ je spojená hranou v G právě tehdy, když je dvojice $f(u), f(v)$ spojená hranou v H. Grafy G a H jsou *izomorfní* pokud mezi nimi existuje izomorfizmus. Zapisujeme $G \cong H$. Viz [Hl07]

Jak je z definice zřejmé, tak izomorfní grafy mají stejný počet vrcholů i hran.
Dále uvádím odhad počtu neizomorfních grafů podle [Ch06]:

Na dané n-prvkové množině $V(n = |V|)$ je $2^{\binom{n}{2}}$ různých grafů, ale těch, které nejsou izomorfní je podstatně méně. Například pro $n=3$ existuje 8 grafů, z toho jen 4 nejsou izomorfní. Je obtížné určit počet neizomorfních grafů, ale jsme schopni stanovit horní $2^{\binom{n}{2}}$ a dolní $\frac{2^{\binom{n}{2}}}{n!}$ odhady.

Abychom se přesvědčili, že funkce neroste rychleji než $2^{\binom{n}{2}}$, odhady zlogaritmujeme a upravíme:

$$log_2\left(2^{\binom{n}{2}}\right) = \binom{n}{2} = \frac{n^2}{2}\left(1 - \frac{1}{n}\right)$$

$$log_2 \frac{2^{\binom{n}{2}}}{n!} = \binom{n}{2} - log_2\, n! \geq \frac{n^2}{2} - \frac{n}{2} - n\, log_2\, n = \frac{n^2}{2}\left(1 - \frac{1}{n} - \frac{2\, log_2\, n}{n}\right) \qquad (3.1)$$

Vidíme, že pro větší n se logaritmy obou funkcí chovají nejhůře jako $\frac{n^2}{2}$, což značí velké množství neizomorfních grafů.

- **Automorfizmus** – je to izomorfizmus grafu na sebe sama a píšeme $G = G'$. Dá se předpokládat, že každý úplný graf, např. K_5, je stejný. (viz [Ch06])

- **Homomorfizmus** – zjednodušeně řečeno se jedná o zobrazení vrcholů na vrcholy a hran na hrany při zachování matice incidence a píšeme $G \sim G'$. (viz [Ch06])

3.2 Identifikace problému izomorfizmu

V kapitole 3.1 byl popsán úvod do izomorfizmu. Problémem izomorfizmu tedy je, zda-li grafy $G=(V, E)$ a $G'=(V', E')$ jsou izomorfní. Přesto, že je definice tohoto problému velmi jednoduchá, je problém nalezení izomorfizmu mezi grafy poměrně složitý. Časová, prostorová a implementační složitost je tedy hlavním problémem u algoritmů, které zjišťují izomorfizmus grafů. Zjišťování izomorfizmu mezi grafy, které mají několik stovek vrcholů, vede k velkým časovým a paměťovým požadavkům na výpočetní systémy. Proto je snaha tyto požadavky snížit. Tento problém spadá do třídy NP, ale nebylo dokázáno, že patří i do třídy NP-úplných. Ale izomorfizmus podgrafů do třídy NP-úplných patří.

Jedním ze způsobů zjištění izomorfizmu, zároveň nejjednodušším, je použití permutace všech vrcholů V a V', což je v podstatě použití tzv. „brutální síly“. Složitost tohoto řešení roste exponenciálně s počtem přibývajících vrcholů. Jsou ale vyvinuty způsoby, jak tuto složitost redukovat. Tyto algoritmy budou popsány níže. Dále zatím nebylo dokázáno, že problém izomorfizmu grafů lze přiřadit do tříd jako P, RP nebo BPP.

Ale jsou speciální případy grafů, u kterých lze problém izomorfizmu zařadit do třídy P:

- **Stromy** – $O(n)$.
- **Rovinné grafy** – $O(n)$.
- **Grafy omezeného stupně obecně**.

V [Ch06] je uvedeno, že lze vhodným algoritmem ukázat polynomiální složitost i pro izomorfizmus dalších grafů, např. plně separabilních nebo kompletních grafů.

V [Wi08] je zmíněno, že při použití metody Monte Carlo je izomorfizmus grafů řešitelný v polynomiálním čase.

Hlavním problémem algoritmů je tedy časová, prostorová a implementační složitost. V následující kapitole rozebírám možná řešení tohoto problému a představuji vhodné algoritmy, které jsem využil při experimentech.

3.3 Návrh řešení problému izomorfizmu

Pro řešení problému izomorfizmu grafů je známo mnoho metod a způsobů, jakým lze izomorfizmus grafů nalézt. V této kapitole předkládám možná řešení tohoto problému. Uvedu vylepšení pro zefektivnění zjišťování izomorfizmu a představím několik algoritmických přístupů k tomuto problému. Hlavními představenými algoritmy jsou Ullmannův algoritmus, VF2 algoritmus a zjišťování pomocí kanonické formy.

3.4 Užitečná vylepšení zjišťování izomorfizmu

Přehledný souhrn vylepšení pro zjišťování izomorfizmu nalezneme v [Ch06]. A následující text z něj bude vycházet.

Ve většině algoritmů se permutuje matice adjacence. To odpovídá prohledávání grafu do šířky nebo hloubky. Jak bylo zmíněno v kapitole 2.4.2, klade matice velké paměťové nároky, proto jde o to, aby se prohledávaný prostor co nejvíce zredukoval. Tj. aby se provádělo co nejméně permutací – nejhůře $n!$. Pokud by nestačila paměť, nelze problém izomorfizmu rozhodnout. Částečným řešením mohou být následující vylepšení algoritmů:

- Využití invariantů – jde o základní krok, je třeba určit, je-li možné, aby byly grafy izomorfní. Při odlišnosti invariantů není třeba procházet celým stavovým prostorem. Důležité je, aby čas nutný k vytvoření invariantů byl v nejhorším případě polynomiální. Nejefektivnějšími invarianty se z tohoto hlediska zdají být skóre grafu a spektrum grafu, jenž je algebraicky odvozeno od matice adjacence.

- Rozklad a vytvoření matice vzdáleností – pomocí tohoto lze rozložit graf na spojité podgrafy (je-li to možné), čím se může omezit počet potřebných permutací. Vhodné je nalezení spojité komponenty grafu pomocí matice dosažitelnosti a hlavně matice vzdáleností.

- Není-li již možné dále graf rozložit, hodí se nalézt některé specifické podgrafy ve vzdálenostní matici. Příkladem specifických podgrafů můžou být cykly C_X nebo kliky K_X. Této metodě se říká slovníková.

- Použití algoritmů vyvinutých v umělé inteligenci. Např. back-tracking, forward-checking nebo bidirectional search.

- Je taky dobré použít vhodnou datovou strukturu grafu přizpůsobenou přímo zvolenému algoritmu.

- Paralelní zpracování algoritmů je taky možné, protože spousta algoritmů využívá mnoho jednoduchých operací, které se dají provádět najednou.

3.5 Algoritmy pro zjišťování izomorfizmu

Algoritmů pro zjišťování izomorfizmu je hodně. Většinou jsou kombinací několika metod a možných vylepšení uvedených v kapitole 3.4. V této části práce se zaměřím především na Ullmannův a VF2 algoritmus pro hledání izomorfizmu v grafech, z toho důvodu, že jsem se v implementační části této práce zabýval právě těmito algoritmy. Dále je blíže popsán algoritmus založený na kanonické formě.

V případě rozsáhlých grafů je třeba rozdělit zpracování do několika fází. Ve fázi tzv. „předzpracování" můžeme provést konstrukci matice adjacence, dosažitelnosti a vzdáleností. Dále je dobré nalézt spojité komponenty a vypočítat jejich invarianty, které jsou potřebné k vyloučení izomorfizmu grafů nebo také k sestavení rozhodovacích funkcí pro algoritmy popsané dále. Můžeme také použít slovníkovou metodu.

Pokud použijeme matici adjacence pro zjištění izomorfizmu, je užitečné ji uspořádat podle vzdálenosti a to tím způsobem, že sousedící vrcholy umístíme v matici vedle sebe (je-li to možné). Tím omezíme později prováděné permutace.

3.5.1 Permutace vrcholů grafu

Tento algoritmus byl již dříve zmíněn. Snaží se najít permutační matici mezi vrcholy procházením všech možných řešení.

Je dostatečně rychlý pro malé nepravidelné grafy s malou mírou automorfismu. Je vhodné použít nějaký typ předzpracování. Například lze využít stupně vrcholů. Permutujeme jen vrcholy se shodným stupněm. Tímto způsobem se dá docela zrychlit algoritmus, ale neřeší případ, kdy má graf málo tříd stupňů vrcholů.

3.5.2 Ullmannův algoritmus

Je jedním z nejpoužívanějších algoritmů pro zjišťování izomorfizmu grafů a také i podgrafů. V roce 1976 jej navrhl **J. R. Ullmann** a v dokumentu [Ull76] jej popisuje. Při následujícím popisu algoritmu jsem vycházel z tohoto zdroje a také z následujících [Ch06], [MB95].

Tento algoritmus je založen na základě tzv. backtrackingu (zpětném vyhledávání, procházení stromu do hloubky) s kombinací forward checkingu (dopředné testování). Využívá maticovou reprezentaci obsahující shody mezi hledaným podgrafem $G=(V, E)$ a grafem $G'=(V', E')$. Rozměr této matice je dán počtem vrcholů $n=|V|$ a $m=|V'|$ a tedy velikost matice je nxm. Matice obsahuje 0 a 1. Matice $M = [m_{ij}]$ $(1< i <|V|, 1< j <|V'|)$ se sestrojuje následovně:

$$m_{ij} = \begin{cases} 1, \textit{pokud je možné zobrazení } v_i \in V \textit{ na } v_j \in V' \\ 0 \textit{ jinak} \end{cases} \tag{3.2}$$

Jinými slovy pokud pro j-tý vrchol grafu G' platí, že má stejný nebo větší počet vstupujících i odchozích hran jako i-tý vrchol grafu G' a mají stejné atributy, potom se na $[i, j]$ pozici matice M_{ij}

zapíše 1, jinak 0. Tento postup sestrojení matice M je pro hledání izomorfizmu podgrafů. Dále uvádím vzorec pro tento postup:

$$m_{ij} = \begin{matrix} 1, pokud\ \deg_+(v_j \in V') \geq \deg_+(v_i \in V) \,\&\, \deg_-(v_j \in V') \geq \deg_-(v_i \in V) \\ 0\ jinak \end{matrix} \tag{3.3}$$

Následující vzorec vyjadřuje postup setrojení matice M pro hledání izomorfizmu grafů:

$$m_{ij} = \begin{matrix} 1, pokud\ \deg_+(v_j \in V') = \deg_+(v_i \in V) \,\&\, \deg_-(v_j \in V') = \deg_-(v_i \in V) \\ 0\ jinak \end{matrix} \tag{3.4}$$

Jednička tedy v matici označuje kompatibilní vrcholy, které splňují podmínku izomorfizmu a aby byl graf izomorfní, je nutné, aby v každém řádku matice M byl alespoň jedna jednička. Pokud v některém z řádků chybí, nejedná se o izomorfizmus a algoritmus se může ukončit. Při procházení stavového prostoru se používá funkce pro forward checking, která má za úkol omezit prohledávaný prostor. Tato funkce je postavena na podmínce takové, že sousedící vrcholy v grafu G, musí být zobrazeny pouze na sousedící vrcholy v grafu G'. Pokud není tato podmínka splněna, je jisté, že není možné nalézt izomorfizmus podgrafu. Dále uvádím pseudokód Ullmannova algoritmu, kde M je předzpracovaná matice, P je množina mapování vrcholů a i je hloubka zanoření algoritmu:

```
ALGORITHM ULLMANN
PROCEDURE Backtrack(P, i, M)
 IF i > n THEN P reprezentuje izomorfizmus podgrafu z G do G_I.
 Vypíše P a vrátí se.
 END IF
 FOREACH m_ij = 1 AND j = 1 TO m
       i.) Nastaví  P = P ∪ {(v_i, w_j)}
       ii.) FOREACH k > i Nastaví M' = M AND m'_kj = 0
             END FOREACH
       iii.) IF Forward_checking(P, i, M') = TRUE THEN
                   CALL Backtrack(P, i + 1, M')
              END IF
       iv.) Odstraň (v_i, w_j) z P
 END FOREACH

PROCEDURE Forward_checking(P, i, M)
 1: FOR k = i + 1 TO n AND j = 1 to m
       IF m_kl = 0 GOTO STEP 1
```

```
        Zkontroluj: IF (v_k, w_l) je konzistentní se všemi
        mapováními v P. Presněji:
           FOREACH (v,w) ∈ P
              i.) IF je hrana e = (v_k, v) ∈ E THEN zde musí být hrana
              e_I = (w_l, w) ∈ E_I  s  v(e) = v_I(e_I)  END IF
              ii.) IF je hrana e_I = (w_l, w) ∈ E_I  THEN zde musí být
              hrana  e = (v_k, v) ∈ E  s   v_I(e_I) = v(e)  END IF
              IF obě podmínky i.) a ii.) jsou true THEN nastav
              p_kl=1 ELSE nastav p_kl=0
              END IF
           END FOREACH
     2: IF existuje řádek k v M takový, že m_kl = 0 FOR l = 1,...,m
     THEN vrať FALSE jinak TRUE
     END IF

PROCEDURE Ullmann
Inicializace matice M
CALL Backtrack(∅, i, M)
```

Tento algoritmus je založen na rekurzi a v každém prohledávaném stavu se ukládá aktuální matice M, což vede k velkým paměťovým narokům u zjišťování izomorfizmu pro velké grafy. Paměťová složitost tohoto algoritmu je $O(N^3)$.

Zajímavé vylepšení algoritmu je uvedeno v [Ic00] a snižuje paměťovou složitost na $O(N^2)$.

3.5.3 Kanonická forma

Při popisu kanonické formy pro hledání izomorfizmů grafů jsem vycházel hlavně z následujících zdrojů [Ch06] a [Kr01].

Hlavní myšlenkou tohoto způsobu nalezení izomorfizmu grafů je převedení grafu na tzv. kanonickou formu a následné porovnání s kanonickou formou jiného grafu. Pokud jsou kanonické formy shodné, je zaručeno, že jsou oba grafy vzájemně izomorfní.

Protože porovnáváme jenom kanonické formy, je nalezení izomorfizmu grafů jednoduché a rychlé. Nevýhodou je, že je třeba vypočítat kanonickou formu, jejíž složitost je v nejhorším případě stejná jako při hledání všech permutací grafu.

Mějme tedy graf $G = (V, E)$ a adjacenční matici $A(G)$ orientovaného grafu, kde a_{ij} jsou prvky této matice. Potom definujeme charakteristický vektor orientovaného grafu jako:

$$\alpha(G) = \{a_{11}, a_{12}, ..., a_{1n}, a_{21}, ..., a_{2n}, ..., a_{nn}\} \quad (3.5)$$

Dále definujeme lexikografické pořadí nad charakteristickými vektory (např. (1,0,0)>(0,0,1)). Kanonickou formou pro graf G je poslední charakteristický vektor v lexikografickém pořadí z množiny všech vhodných charakteristických vektorů. Permutací pořadí vrcholů každého grafu G lze obvykle získat spoustu charakteristických vektorů tohoto grafu. Mějme P_n množinu všech permutací posloupností s prvky(vrcholy) $n = |V|$. Potom můžeme kanonickou formu zapsat jako:

$$Kannon(G) = \max \alpha(G^{P_n}) \quad (3.6)$$

Jak již bylo zmíněno, tak tento způsob má stejnou složitost jako při hledání všech permutací. Je tedy lepší použít chytřejší přístup. Příkladem může být postup uvedený v [Kr01]. Ve zkratce tento algoritmus generuje jenom nepřiřazené vrcholy a ten s „největším“ značením přidá do vektoru kanonické formy. Toto vylepšení má složitost podobnou jako Ullmannův algoritmus, ale u některých typů grafů je Ullmannův algoritmus lepší. Pro velké grafy je vhodné použít nějaké předzpracování. Možné řešení je uvedeno v např. v [Mc05].

3.5.4 VF2 algoritmus

Při popisu tohoto algoritmu jsem čerpal z následujícího zdroje [CFSV01]. Jedná se o algoritmus navržený speciálně pro vyhledávání izomorfizmu velkých grafů nebo podgrafů a přidává vylepšení k VF algoritmu uvedeném v [CFSV99]. Hlavním vylepšením tohoto algoritmu oproti VF algoritmu je, že zkoumání prohledávaného prostoru je uspořádáno takovým způsobem, aby výrazně snížilo paměťové nároky.

Algoritmus je představen pro orientované grafy, ale lze jej jednoduše upravit pro hledání v neorientovaných grafech. Algoritmus využívá deterministické prohledávání grafu do hloubky. Vyhledávací část algoritmu lze popsat ve smyslu SSR (State Space Representation), tj. v překladu jako stavová prostorová reprezentace. Každému stavu *s*, vyhledávacího procesu, můžeme přiřadit částečné řešení mapování *M(s)*, které obsahuje pouze podmnožinu části mapovací funkce *M*. Částečné mapovací řešení jednoznačně určí dva podgrafy z G_1 a G_2 , označené $G_1(s)$ a $G_2(s)$, které se získají způsobem takovým, že se vyberou vrcholy z G_1 a G_2 právě takové, jenž jsou obsaženy v částech $M(s)$ a jsou spojeny větvemi. Následnuje zjednodušený pseudokód algoritmu z [CFSV01]:

```
ALGORITHM VF2
PROCEDURE Match(s)
VSTUP: přechodný stav s; počáteční stav s0 kde M(s0) = ϕ M(s0)=∅
VÝSTUP: mapování mezi dvěma grafy
IF M(s) pokrývá všechny vrcholy grafu G2 THEN OUTPUT M(s)
ELSE Vypočítej množinu možných párů P(s) pro zahrnutí do M(s)
```

```
    FOREACH (n,m) ∈ P(s)
        IF F(s,n,m) THEN
            Zpracuj stav s' získaný přidáním (n,m) do M(s)
            CALL Match(s')
        END IF
    END FOREACH
    Obnovení předchozích hodnot polí při návratu z rekurze
END IF
END PROCEDURE
```

Tato funkce je volána rekurzivně. *P(s)* je zde množina možných párů v daném stavu. Funkce $F(s,n,m)$ je zde použita pro omezení prohledávacího stromu. Pokud vrací *true*, je garantováno, že stav *s'* získaný přidáním (n,m) do *s*, je částečným izomorfizmem, pokud stav *s* je izomorfizmem. A tedy konečný stav je buďto izomorfizmem mezi G_1 a G_2 nebo izomorfizmem podgrafu mezi podgrafem G_1 a G_2. Navíc funkce *F* odstraňuje stavy, které nevedou ke kompletnímu řešení.

Aby mohl být algoritmus použit na rozměrné grafy, využívá šesti polí, kde velikost tří polí je počet vrcholů grafu G_1 a další tři pole mají velikost odpovídající počtu vrcholů grafu G_2. Ve dvou polích z obou grafů se uchovává aktuální mapování vrcholů. Zbývající čtyři pole tvoří množinu možných párů *P(s)* a uchovávají aktuální hodnotu hloubky v SSR stromu náležející stavu, ve kterém vstoupil vrchol do odpovídající množiny. Těchto šest polí je sdíleno ve všech stavech rekurze. Tím je dáno, že paměťová náročnost je dána počtem vrcholů obou grafů a prohledávání grafu do hloubky zaručuje, že v paměti může být nanejvýš *N* stavů v jednom čase, kde *N* je počtem vrcholů. A tedy paměťová složitost tohoto algoritmu je *O(N)*.

3.5.5 Další algoritmy

Algoritmus založený na statistických vlastnostech pomocí metody **Monte Carlo** je uveden v [Ga87]. V dokumentu [MB95] je uveden algoritmus pro hledání izomorfizmu v polynomiálním čase. Tento algoritmus je založen na předzpracování, ve kterém používá modelové grafy pro vytvoření rozhodovacího stromu.

Další algoritmy můžeme najít na odborných konferencích, jako je např. ACM (http://portal.acm.org).

4 Správa grafů v databázi

Grafů se dá využít v různých vědních a technických odvětvích. Pro uložení údajů v podobě grafů je výhodné využít nějaký databázový systém. Příkladem mohou být geografické systémy, nějaké CAD/CAM systémy, www stránky, systémy v chemii pro nalézání nových sloučenin, v počítačovém vidění, bioinformatice apod.

Pro správu grafů v databázi je třeba definovat strukturu dat, která bude uložena v databázi, integritní omezení a operace spadající do dotazovacího jazyka, které umožní práci s těmito daty. Tyto náležitosti zaštiťuje jako celek pojmenování databázový model (viz [AG05]).

- **Struktura dat** – závisí na objektech a vztazích mezi nimi, které modelujeme pomocí grafů. V případě, že chceme použít nějakou specifickou operaci jako například zjištění izomorfizmu, je vhodné pro zvolený algoritmus upravit strukturu tak, aby byl algoritmus co nejefektivnější.
- **Operace** – v databázi grafů mohou být např. operace jako přidání a mazání vrcholů a hran, vyhledávání nejkratší cesty, získání sousedů daného vrcholu, získání podgrafu a podobné.
- **Integritní omezení** – zajišťují konzistenci dat, která jsou přímo spjata se strukturou grafu. Příkladem může být pojmenování vrcholů unikátními jmény nebo při odstranění, popř. úpravě, nějakého vrcholu je třeba, aby se tyto změny promítly i do dat reprezentující hrany grafu.

Dále uvádím několik modelů pro správu grafů v databázi:

- **Logický datový model (LDM)** – je to úplný databázový model (zahrnuje datové struktury, dotazovací jazyk a integritní omezení). Zevšeobecňuje relační, hierarchické a síťové modely. Model popisuje mechanismy pro přeuspořádání dat a logický a algebraický dotazovací jazyk. Více podrobností v [KV84].
- **Model Hypernode** – model založený na datové struktuře hypernode. Hypernode umožňuje vnořovat grafy. Může být použit pro reprezentaci jednoduchých a složitých objektů, stejně jako pro mapování a záznamy. Klíčovým rysem je schopnost zapouzdřit informaci. Je to úplný databázový model. Více viz [LP90].
- **Model GOOD** – tento model je hlavně navržen pro vývoj databázových grafických prostředí koncového uživatele. Schéma a instance jsou zde reprezentovány orientovanými pojmenovanými grafy a manipulace s daty je zde vyjádřena grafovými transformacemi. Model umožňuje pouze dva typy uzlů a to tisknutelné a netisknutelné. Umožňuje také jen dva typy hran a to funkční (mají jedinečnou hodnotu) a nefunkční (mají více hodnot). Model

zahrnuje datový transformační jazyk s grafickou syntaxí a sémantikou. Obsahuje pět základních grafových transformací. Čtyřmi operacemi jsou přidávání a mazání vrcholů a hran. Pátá operace zvaná abstrakce je použita pro seskupování vrcholů na základě funkčních nebo nefunkčních vlastností. Model zahrnuje také např. makra. Více viz [GPB94].

- **Model Gram** – model, kde jsou data organizována jako graf. Schéma je zde orientovaný pojmenovaný multigraf, kde má každý uzel jméno určitého typu, který má přiřazený určitý okruh hodnot. Analogicky má hrana pojmenování, které reprezentuje vztah mezi typy. Rysem Gram modelu je použití speciálních objektů pro jasnou definici cest zvanou procházky. Dotazovací model navrhuje algebraický jazyk založený na regulárních výrazech. Gram model nedefinuje integritní omezení. Více viz [AS92].

Podrobnější všeobecný přehled těchto modelů lze najít v [AG05]. Nalezneme zde i další modely jako např. GGL, GROOVY, GMOD, PaMaL, GOAL a GDM. V odkazech uvedených přímo u příslušného modelu jsou odkazy, které pojednávají přímo o daném modelu.

Tato práce je zaměřena na graf odkazů v internetové encyklopedii Wikipedia. Vrcholy zde reprezentují jednotlivé stránky a orientované hrany jsou odkazy mezi nimi. Tento graf má 151817 vrcholů a 3330819 hran.

5 Implementační část

V této části popíši jak jsem postupoval při návrhu a tvorbě jednotlivých tříd, které jsou součástí této práce. Uvedu způsob reprezentace grafu, implementace vybraných algoritmů, mezi něž patří VF2 algoritmus a Ullmannův algoritmus.

5.1 Implementační prostředí

Pro implementaci algoritmů pro hledání izomorfizmu v grafech jsem vybral programovací jazyk Java. Důvodem tohoto výběru je snadná přenositelnost a také fakt, že většina algoritmů pro nalezení izomorfizmů v grafech je implementována v jazyce C nebo C++ a implementace těchto algoritmů v Javě by byla větším přínosem. Pro vývoj bylo použito prostředí NetBeans ve verzi 6. Pro reprezentaci databáze pro lokální kopii české Wikipedie byla vybrána databáze MySQL. Tato databáze je uložna na výpočetním serveru Athena1 a na totožném serveru běžely experimentální výpočty. Počítačová konfigurace serveru Athena1 je následující:

Procesor: 2xAMD Opteron 2,8GHz

Paměť: 16GB RAM

5.2 Reprezentace grafu

V části o reprezentaci grafu jsou popsány různé možnosti reprezentace grafů v počítačích. Já jsem několik těchto možností implementoval. Navrhl jsem abstraktní strukturu grafu, která definuje následující metody pro práci s grafem. Jedná se o přidávání vrcholů a hran, odstranění vrcholů a hran, získání počtu vrcholů a hran, získání počtu vstupujících nebo odchozích hran v daném vrcholu, získání vstupujících nebo odchozích sousedů v daném vrcholu, zjištění, zda-li existuje zadaná hrana. Tuto abstraktní třídu jsem zdědil a vyvořil třídy pro reprezentaci grafu pomocí adjacenčního seznamu, adjacenční matice a grafu uloženého v databázi.

Protože v Javě nelze mít pole s velikostí prvku jeden bit, ale minimálně jeden byte (záleží na JVM), vytvořil jsem vlastní strukturu dvourozměrného pole pro ukládání bitových hodnot. Tato struktura používá klasické dvourozměrné pole a ukládá do něj čísla o velikosti short, tj. 16bitové číslo. V každé buňce je tedy uloženo 16bitů a k nim se přitupuje pomocí bitových operací. Důvodem k vytvoření takovéto struktury bylo to, že měly experimenty probíhat na grafu, který má kolem 150000 vrcholů. Takto velké grafy zabírají spoustu paměti a navíc by Ullmannovu algoritmu brzy došla paměť při hledání izomorfizmu podgrafů. Paměťová složitost Ullmannova algoritmu je $O(N^3)$. A tedy i tento algoritmus využívá matice s bitovými hodnotami.

Je jistě zřejmé, ze graf reprezentovaný adjacenční maticí nebo seznamem využívá datových struktur jako je pole nebo seznam (v mém případě Vector). U reprezentace grafu v databázi je však nutné zajistit přístup k databázi s grafem, provést namapování vrcholů na vrcholy uložené v databázi, zajistit, aby data v databázi byla konzistentní, tj. aby nebyly v databázi hrany mezi vrcholy, které neexistují. Dále je vhodné přidat indexy na sloupce, podle kterých budeme hledat v databázi. Výrazně to urychlí vyhledávání.

5.3 Struktura grafu v databázi

Podstatná část této práce se zabývá grafem uloženým v databázi. V kapitole o možné reprezentaci grafu v databázi je zmíněno několik způsobů jak ukládat graf do databáze. Jako zdroj grafu jsem použil grafy odkazů lokální kopie české Wikipedie. Vrchol zde reprezentují jednotlivé stránky wikipedie a orientovaná hrana je zde odkaz ze stránky na jinou stránku. Tabulku reprezentující jednotlivé stránky (vrcholy) jsem ponechal, avšak tabulka pro ukládání odkazů stránek byla nevyhovující, proto jsem vytvořil vlastní tabulku se dvěma sloupci. Jeden sloupec pro vrchol, ze kterého hrana vychází a druhý sloupec, do nějž hrana vstupuje. Na sloupcích jsem vytvořil indexy pro rychlejší vyhledávání v tabulkách databáze. Tento způsob uložení grafu v databázi byl vybrán z toho důvodu, že je jednoduchý a neklade velké nároky na dotazovací jazyk. Graf databáze obsahuje 151817 vrcholů a 3330819 hran. Bylo nutné odstranit některé hrany, které porušovaly konzistenci grafu.

5.4 Implementované algoritmy

Nelze jednoznačně určit, který algoritmus je nejlepší, protože záleží na typu a velikosti grafů. Pro implementaci algoritmu zjišťování izomorfizmu grafů jsem vybral Ullmannův algoritmus, který je v současné době asi nejvíce využíván. A dále jsem vybral VF2 algoritmus, který je navržen speciálně pro velké grafy a tedy se hodí pro experimenty prováděné na grafu odkazů české Wikipedie, protože počet vrcholů tohoto grafu je kolem 150 tisíc.

5.4.1 Ullmannův algoritmus

Ullmannův algoritmus je popsán v části zabývající se algoritmy pro hledání izomorfizmu v grafech. Pro malé grafy je tento algoritmus vhodný. Ale u rozsáhlých grafů již spotřebuje hodně paměti. Je to dáno tím, že používá matici o velikosti násobku vrcholů obou porovnávaných grafů. A při rekurzivním volání tohoto algoritmu se dělá kopie této matice v daného stavu. Toto je velké množství spotřebované paměti $O(N^3)$. Časová náročnost se u tohoto algoritmu zvyšuje především při inicializaci tohoto algoritmu, kdy je třeba sestrojit matici pro tento algoritmus. Další zpozdění je zapříčiněno vytvářením kopie matice v každém stavu rekurze. V experimentální části této práce jsou

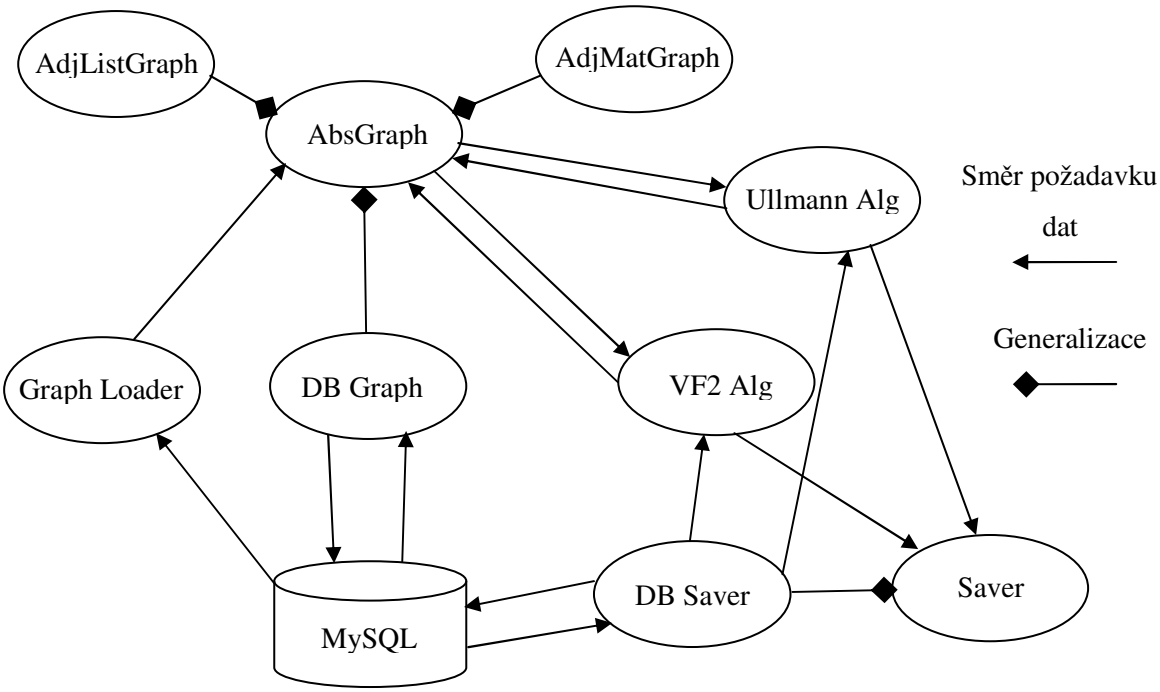
popsány výsledky tohoto algoritmu na velkém grafu. Pro zajímavost jsem implementoval tento algoritmus i s mnou implementovanou bitovou maticí.

5.4.2 VF2 algoritmus

Hlavním problémem v této práci byl velký graf reprezentovaný grafem odkazů v databázi Wikipedie. Což klade velké nároky na použitý algoritmus. V začátcích této práce jsem vybral Ullmannův algoritmus pro zjišťování izomorfizmu grafů z toho důvodu, že je zřejmě nejznámější a hojně využíván. Ale pro hledání velkých grafů byl nevyhovyjící a proto jsem hledal alternativu, kterou jsem našel v algoritmu VF2. Tento algoritmus nedělá žádné předzpracování jako se dělá v Ullmannově algoritmu. A také nevytváří v každém stavu rekurze kopie dat potřebných pro nalezení izomorfizmu. A tedy neklade přílišné nároky na paměť, což je pro hledání velkých grafů žádané. V části zabývající se experimenty je tento algoritmus srovnán s Ullamnnovým algoritmem.

5.5 Další implementované části

Pro experimentování jsem naimplementoval třídu pro ukládání nalezených izomorfizmů podgrafů do databáze. Tato třída ukládá data v novém vlákně, takže algoritmus pro hledání izomorfizmů podgrafů může běžet dál a není brzděn ukládáním výsledků. Další implementovanou třídou je načítání grafu z databáze do paměti.



Obrázek 5.1: Schéma navržených tříd

Na obrázku 5.1 je schéma navržených tříd. V oválech jsou třídy, jednoduchá šipka ukazuje směr jakým jdou požadavky na data a čára se čtvercem představuje generalizaci (ukazuje na abstraktní třídu).

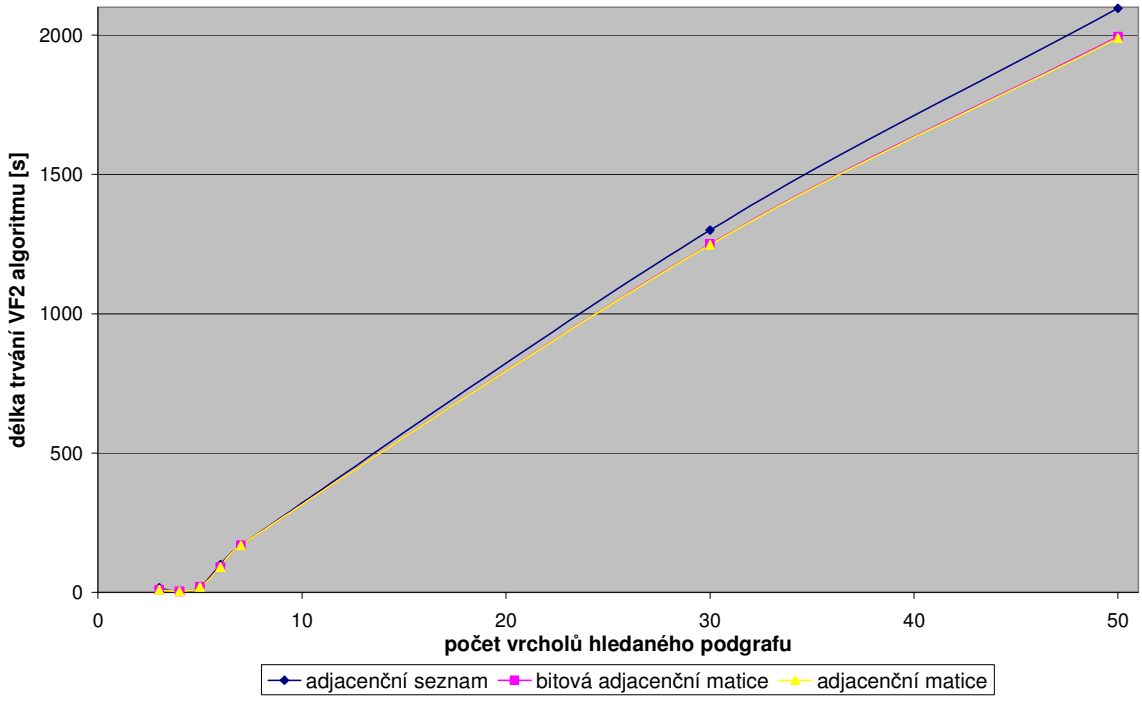
6 Experimetální část

Na implementovaných algoritmech jsem provedl několik experimentů. V této kapitole tyto experimenty teoreticky rozeberu, popíši jejich výsledky a srovnám s již publikovanými experimenty v oblasti hledání izomorfizmu v grafech.

6.1 Srovnání časové náročnosti struktur pro reprezentaci grafu

Cílem tohoto experimentu pro mne bylo zjistit, která z mnou implementovaných struktur pro reprezentaci grafu má nejrychlejší přístup ke svým položkám. Testoval jsem tři reprezentace grafu a to reprezentaci pomocí matice adjacence, seznamu adjacence a mé vlastní implementované bitové matice. Teoretickým předpokladem pro tento experiment bylo, že nejpomalejší přístup by měl být k prvkům v seznamu adjacence a nejrychlejší k prvkům matice adjacence. Bitová matice by měla být mezi těmito dvěma.

Pro tento experiment jsem použil algoritmus VF2, protože Ullmannův algoritmus se dlouhou dobu inicializuje a zbytečně by to zdržovalo testování. Při testu jednotlivých struktur jsem pro danou strukturu vytvořil graf o velikosti v řádech desítek vrcholů a tento graf vyhledával pomocí VF2 algoritmu jako podgraf grafu, který je tvořen databází.



Obrázek 7.1: Časová náročnost struktur

Na obrázku 7.1 je vidět graf provedeného experimentu. Z něj vyplývá, že při použití testovaných struktur pro reprezentaci malých grafů jsou na tom v podstatě všechny struktury podobně. Adjacenční matice a adjacenční bitová matice na tom byly víceméně stejně u všech testovaných grafů. Adjacenční seznam začal za těmito dvěmi maticemi zaostávat u grafů s vyšším počtem vrcholů. Určitě by bylo dobré udělat experimenty pro grafy, které mají vrcholy v řádech stovek až tisíců. Vzhledem k tomu, že pro tak velké grafy by algoritmus pro vyhledávání izomorfizmu podgrafů běžel několik dní, ne-li týdnů, nebylo v mých silách toto otestovat. I proto, že bylo třeba udělat další experimenty.

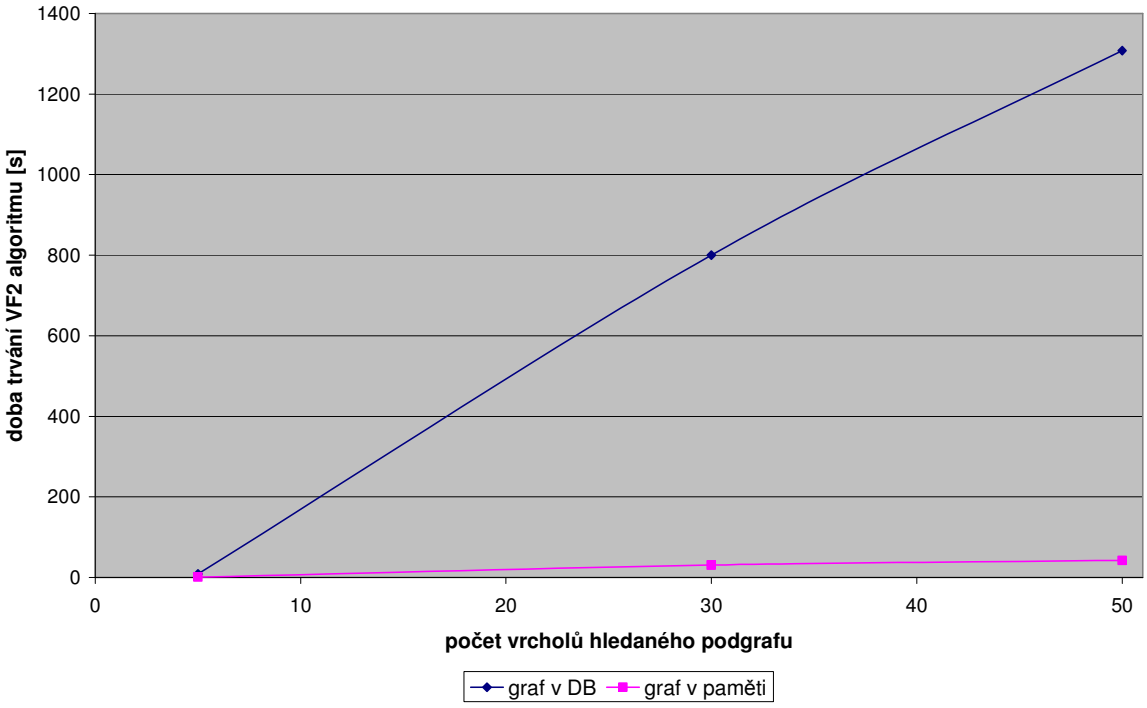
6.2 Srovnání vyhledávání izomorfizmu v paměti a v databázi

Tímto experimentem jsem chtěl zjistit, jestli je vyhledávání izomorfizmu v grafech rychlejší, když máme graf uložený v paměti a nebo v databázi. A jestli je vůbec možné velký graf načíst do paměti a jestli je rozumné pracovat s daty pouze v databázi. Teoretickým předpokladem je, že vyhledávání izomorfizmu v grafu uloženém v paměti bude rychlejší, než-li graf uložený v databázi. Dalším předpokladem je to, že graf o velikosti kolem 150000 vrcholů lze načíst do paměti za předpokladu, že se jedná buďto o řídký graf reprezentovaný adjacenčním seznamem a nebo je tento graf načten do bitové matice adjacence. Při použití bitové matice se předpokládá, že tato reprezentace spotřebuje cca 2,8GB operační paměti. Což by mělo dostačovat, vzhledem k tomu, že na výpočetným serveru Athena1 je k dispozici 16GB operační paměti. Z paměťových důvodů jsem opět při tomto experimetu využil algoritmu VF2.

Při experimentu ale nastaly potíže s pamětí, i když jsem předpokládal, že s JVM (Java Virtual Machine) ve verzi 1.5 nenastanou. Toto však nebylo zapříčiněno špatným odhadem potřebné paměti, nýbrž faktem, že JVM mi neumožnil alokovat více paměti než 2,5GB. Z tohoto důvodu jsem byl nucen omezit počet vrcholů v databázi na 100000, které se již pohodlně vešly do 2,5GB.

Na obrázku 7.2 je znázorněn graf ukazující dobu trvání vyhledání izomorfizmu podgrafu v grafu tvořenou databází o velikosti 100000 vrcholů. Fialová čára zde zobrazuje vyhledávání izomorfizmu v grafu nahraného z databáze do paměti a modrá čára reprezentuje vyhledávání izomorfizmu grafu v databázi. Doba potřebná pro nahrání celého grafu do paměti byla kolem 19 minut. Z grafu jasně vyplývá, že vyhledávání izomorfizmu přímo v paměti je několika násobně rychlejší než vyhledávání v databázi. Je vidět, že při zvyšování počtu vrcholů vyhledávaného podgrafu u paměti roste doba trvání jen mírně, kdežto v případě databáze tato doba rapidně narůstá. Takže se předpoklad, že bude paměť rychlejší, potvrdil. U malých podgrafů byla paměť asi 23x rychlejší než databáze a u podgrafů v řádech desítek vrcholů byl rozdíl třicetinásobný. Ale paměťové prostředky (operační paměť) jsou hodně omezené, takže u grafů kolem 400 tisíc vrcholů a výš

narážíme na problém s pamětí a je třeba graf uložit jinším způsobem. Například do již zmíněné databáze.



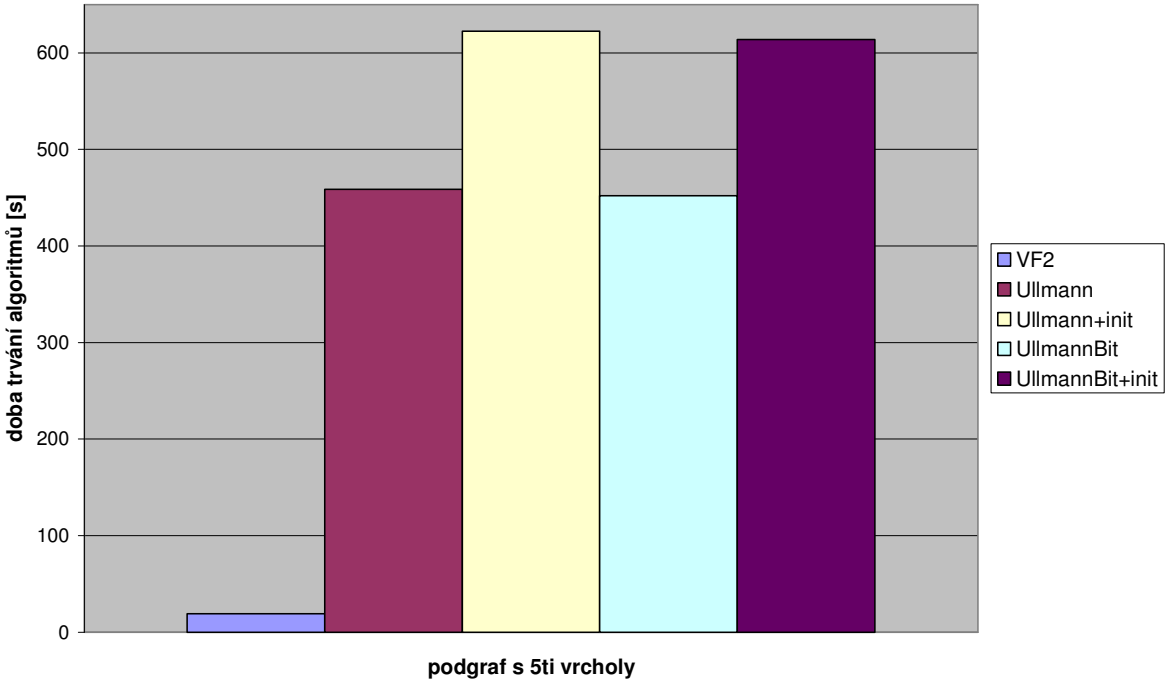
Obrázek 7.2: Experiment časové náročnosti

6.3 Porovnání Ullmannova a VF2 algoritmu

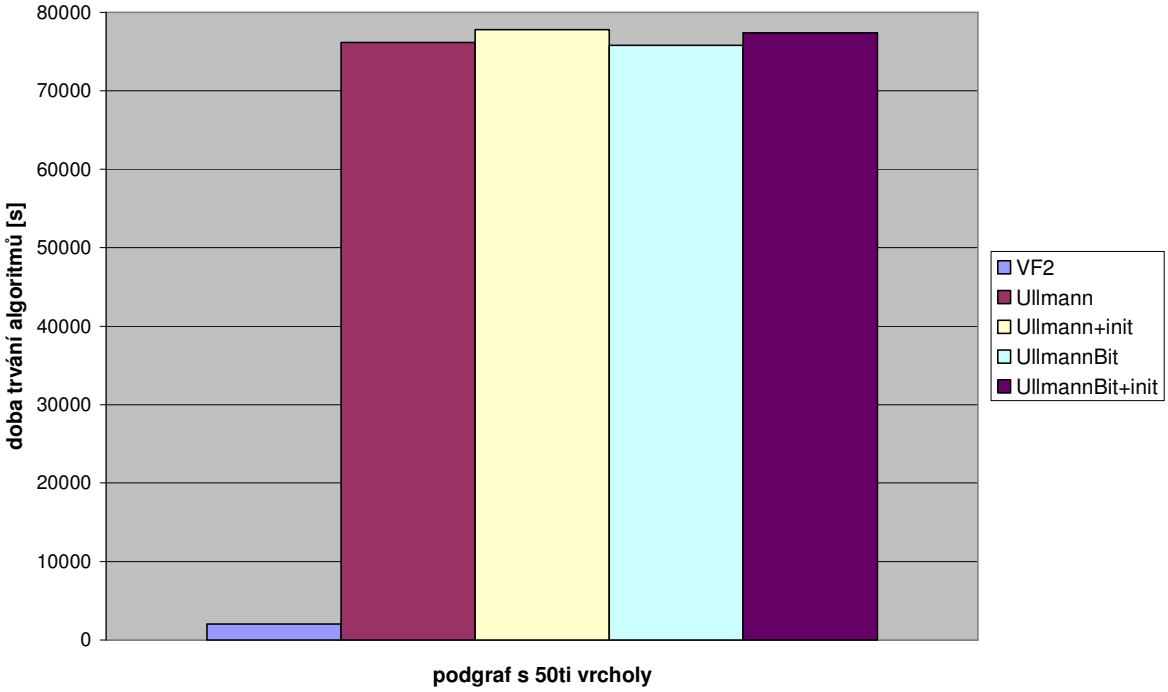
Tento experiment srovnává dva implementované algoritmy. A to Ullmannův a VF2 algoritmus. Testování probíhalo na velkém grafu uloženém v databázi. Předpokladem zde bylo, že by měl být VF2 algoritmus lepší, než Ullmannův algoritmus. Důvodem tohoto předpokladu je fakt, že VF2 algoritmus je navržen na zjišťování izomorfizmu pro velké grafy a experimenty byly prováděny na velmi rozsáhlém grafu.

U tohoto experimentu jsem provedl více testů. Výsledek prvního z nich lze vidět na grafu znázorněném na obrázku 7.3. Testoval jsem zde VF2 algoritmus, Ullmannův algoritmus a Ullmannův algoritmus používající bitovou matici. Vyhledával jsem v databázovém grafu podgraf s pěti vrcholy. Jak je vidět, tak VF2 algoritmus výkonostně jednoznačně překonal obě verze Ullmannova algoritmu. Ullmannův algoritmus s bitovou maticí byl o něco lepší než základní Ullmannův algoritmus. V grafu jsou pro srovnání uvedeny časy Ullmannových algoritmů s i bez předzpracování, které tento algoritmus používá. Obdobný test proběhl i pro podgraf s 50ti vrcholy. Výsledek je možné vidět na obrázku 7.4. Zde už je vidět propastný rozdíl mezi VF2 algoritmem a Ullmannovými algoritmy. U velkých grafů Ullmannův algoritmus zaostává. VF2 algoritmus zde trval zhruba 34 minut, kdežto Ullmannův přibližně 21,5 hodiny. Další test proběhl opět na pěti vrcholech podgrafu. Tentokrát jsem nechal vyhledat v databázovém grafu tento podgraf pětkrát. Výsledný graf je na obrázku 7.5. Zde se

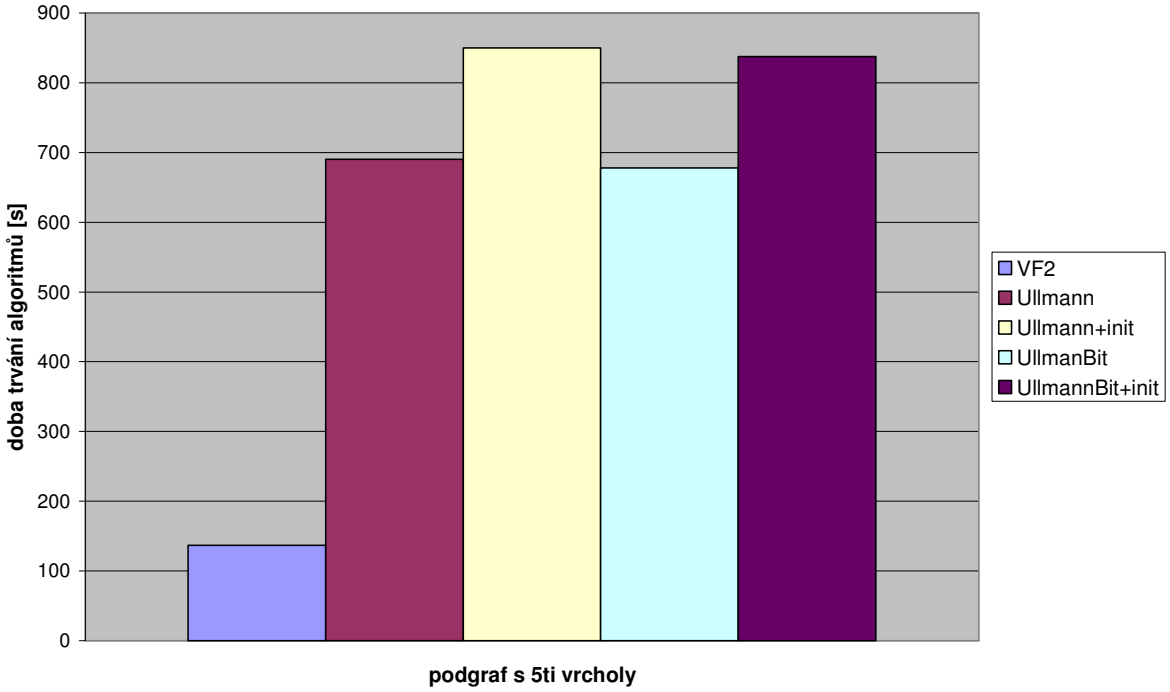
Ullmannův algoritmus o něco zlepšil oproti předchozích testech, ale VF2 algoritmus se mu překonat nepodařilo. Na posledním obrázku 7.6 je graf znázorňující čas potřebný k inicializaci Ullmannova algoritmu. Z grafu lze vyčíst, že s růstem počtu vrcholů hledaného podgrafu roste čas potřebný k inicializaci tohoto algoritmu lineárně.



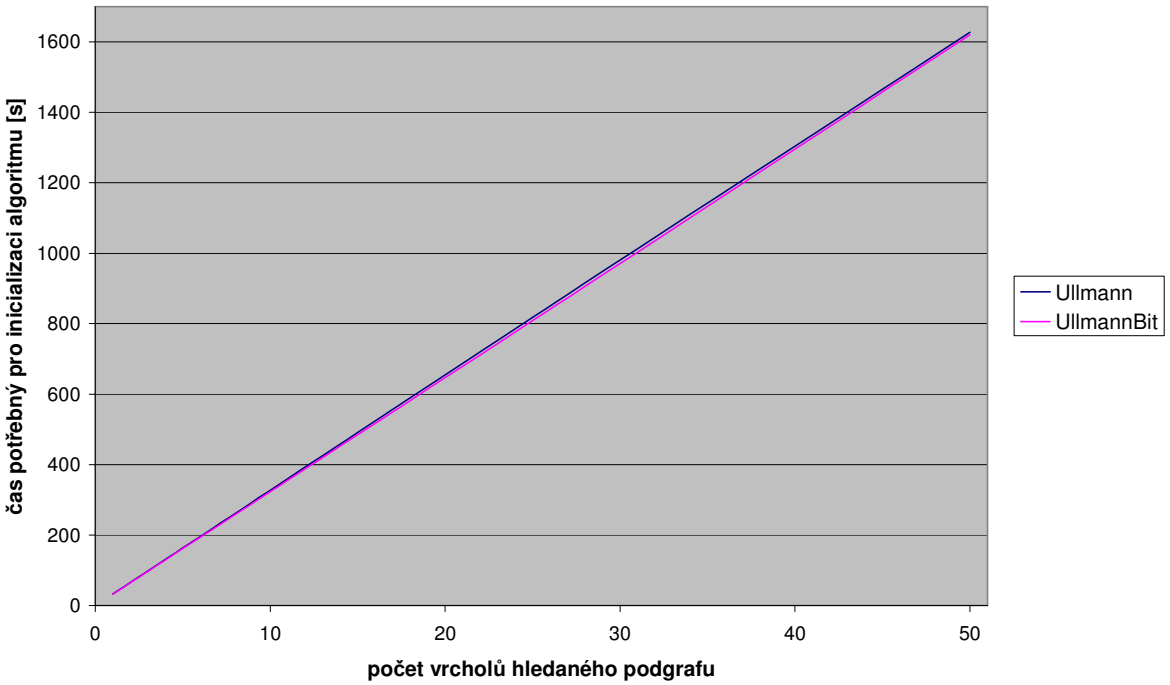
Obrázek 7.3: Srovnání algoritmů pro podgraf s 5ti vrcholy



Obrázek 7.4: Srovnání algoritmů pro podgraf s 50ti vrcholy



Obrázek 7.5: Srovnání algoritmů pro podgraf s 5ti vrcholy (5 nalezených izomorfizmů)



Obrázek 7.6: Čas potřebný k inicializaci Ullmannova algoritmu

6.4 Vyhodnocení Experimentů

Výsledky experimentů dopadly podle teoretických předpokladů. Nicméně jsem byl příjemně překvapen výsledky VF2 algoritmu. Předpokládaná paměťová složitost tohoto algoritmu je *O(N)* a výsledky tuto složitost potvrdily. V [CFSV01] se sice časová složitost jejich VF2 algoritmu neuvádí, ale tento algoritmus byl několikanásobně lepší než Ullmanův. Experimenty také potvrdily, že Ullmannův algoritmus pro grafy s velkým počtem vrcholů nezvládá hledat izomorfizmy v rozumném čase. Dále tyto testy v určité míře potvrdily experimenty prováděné s těmito algoritmy, které jsou uvedené v dokumentu [FSV01].

Až na pár vyjímek jsem vyhledával pouze jeden izomorfizmus podgrafu v databázovém grafu. Bylo to z toho důvodu, že pro nalezení více izomorfizmů v grafu by bylo třeba času v řádech týdnů, možná i měsíců. To závisí od toho jak velké podgrafy se mají hledat. Zkoušel jsem vyhledat všechny izomorfizmy podgrafu o pěti vrcholech v databázovém grafu, ale experiment běžel příliž dlouho (cca 6 dní), proto jsem ho musel zastavit, abych stihl provést i další testy. Za tu dobu co experiment běžel, našel VF2 algoritmus kolem 44 tisíc izomorfizmů.

Protože vyhledávání izomorfizmů trvá u velkých grafů dlouho, je stále ještě mnoho experimentů, které by bylo možné provádět, jen jsou časově velmi náročné.

7 Závěr

Práce se zabývala teorií grafů. Seznámila s její historií a teoretickou částí. Uvedla problémy teorie grafů. Hlavní důraz byl kladen na problém izomorfizmu grafů. Byly zde identifikovány problémy izomorfizmu a návrh jeho možných řešení. Popsal jsem několik algoritmů pro zjišťování izomorfizmu grafů a podgrafů, mezi něž patří VF2 algoritmus, Ullmannův algoritmus a užití kanonické formy. Dále byla popsána správa grafů v databázi a možné reprezentace grafu v programech. Zmínil jsem se o implementaci tříd důležitých pro experimentování s hledáním izomorfizmu podgrafů v grafu uloženého v databázi MySQL. Na závěr jsem představil experimenty, které jsem provedl na grafu odkazů lokální kopie české Wikipedie.

Výsledky experimentů částečně podpořily výstupy experimentů provedených v [FSV01]. Ne však úplně, protože mé experimenty nebyly tak rozsáhlé. A to z časových důvodů. A také protože mé testování bylo zaměřeno hlavně na velký graf v databázi a hledání izomorfních podgrafů tohoto grafu. I přes jinší způsob prováděných experimentů bylo výsledné porovnání VF2 algoritmu a Ullmannova algoritmu odpovídající experimentům v [FSV01].

Izomorfizmus grafu je NP-úplný problém, ačkoliv bylo experimentálně prokázáno, že je v naprosté většině případů řešitelný v polynomiálním čase $O(N^3)$ pomoví Ullmanova algoritmu, ale složitost VF2 je ještě lepší, což dává možnost, že existuje exvivalence třídy $NP = P$ spíše, než naopak. Tato práce navíc prezentuje experimenty provedené na NP-těžkém problému izomorfizmu grafů, který je opět řešitelný v polynomiálním čase, byť nalezení všech izomorfních podgrafů trvá i tak velmi dlouho.

Přínosy této práce vidím v implementaci algoritmů pro zjišťování izomorfizmu grafů v jazyce Java, neboť není mnoho pokusů o implementací těchto algoritmů v tomto jazyce. Dalším přínosem jsou provedené experimenty, které potvrdily teoretické předpoklady a také výsledky experimentů lidí, kteří se tímto tématem zabývali.

V rámci této práce nebylo možné prozkoumat všechna zákoutí týkající se izomorfizmu grafů. Proto by bylo dobré navázat na tuto práci a ještě více nahlédnout do tajů izomorfizmu. Možná další pokračování v této práci bych viděl v implementování i nějakých netradičních metod pro zjišťování izomorfizmu a v rozsáhlém testování izomorfizmu grafů na širokém spektru grafů. Toto rozsáhlé testování je velmi časově náročné a určitě by bylo velkým přínosem poznatků o izomorfizmu grafů. Přínosem by také bylo vyřešení problému s alokací paměti u JVM. Největším přínosem by však bylo přijít s novým originálním způsobem jak hledat izomorfizmus grafů.

Na závěr bych chtěl říct, že izomorfizmus grafů je velmi zajímavé téma pro zkoumání, jen jsem neměl takové časové možnosti, abych toto téma více probádal.

Literatura

[AG05] ANGELS, R., GUTIÉRREZ, C.: Survey of graph database models. *TR/DCC2005-10*. Computer Science Department Technical Report [online]. Chile, 2005. 65s. Dostupný z: http://www.dcc.uchile.cl/~rangles/research/publications/tr-dcc-2005-10.pdf, (květen 2008)

[AS92] AMANN, Bernd, SCHOLL, Michel.: Gram: A Graph Data Model and Query Language. *ECHT '92: Proceedings of the ACM conference on Hypertext*. 1992. str. 201-211. ISBN 0-89791-547-X. Dostupný z: http://doi.acm.org/10.1145/168466.168527, (květen 2008)

[BM82] BONDY, J. A., MURTY U. S. R.: *Graph Theory With Applications*. 5th Edition. New York: Macmillan Press, 1982. 264s. ISBN 0-444-19451-7.

[CFSV99] CORDELLA, Luigi P., FOGGIA, Pasquale, SANSONE, Carlo, VENTO, Mario.: Performance evaluation of the VF Graph Matching Algorithm. *Proc. of the 10th ICIAP*. IEEE Computer Society Press [online]. 1999. str. 1172-1177. Dostupný z: http://amalfi.dis.unina.it/people/vento/lavori/iciap99.pdf, (květen 2008)

[CFSV01] CORDELLA, Luigi P., FOGGIA, Pasquale, SANSONE, Carlo, VENTO, Mario.: An improved algorithm for matching large graphs. *Proc. of the 3rd IAPR TC-15 Workshop on Graphbased Representations in Pattern Recognitionl* [online]. 2001. str. 149-159. Dostupný z: http://amalfi.dis.unina.it/graph/db/papers/vf-algorithm.pdf, (květen 2008)

[FSV01] FOGGIA, Pasquale, SANSONE, Carlo, VENTO, Mario.: A Performance Comparsion of Five Algorithms for Graph Isomorphism. *Workshop on Graph-based Representations in Pattern Recognition* [online]. 2001. Dostupný z: http://amalfi.dis.unina.it/people/vento/lavori/gbr01bm.pdf, (květen 2008)

[Ga87] GALIL, Zvi et al. An O(n3log n) deterministic and an O(n3) Las Vegas isomorphism test for trivalent graphs. *Journal of the ACM* [online]. Vol. 34, Issue 3, 1987. Dostupný z: http://portal.acm.org/citation.cfm?doid=28869.28870, (květen 2008)

[GPB94] GYSSENS, M., PAREDEANS, J., BUSSCHE, J. Van Den, GUCHT, D. Van.: A Graph-Oriented Object Database Model. *Journal of IEEE Transactions on Knowledge and Data Engineering* [online]. 1994. ISSN 1041-4347. Dostupný z: http://doi.ieeecomputersociety.org/10.1109/69.298174, (květen 2008)

[GZM98] ZHUGE, Y., GARCIA-MOLINA, H.: Graph Structured Views and Their Incremental Maintenance. *ICDE '98: Proceedings of the Fourteenth International Conference on Data Engineering*. 1998. str. 116-125. ISBN 0-8186-8289-2. Dostupný z: http://doi.ieeecomputersociety.org/10.1109/ICDE.1998.655767, (květen 2008)

[Hl07] HLINĚNÝ, Petr.: *Teorie grafů*. FI MU Brno, 2007. 111s. Dostupný z: http://www.fi.muni.cz/~hlineny/Vyuka/GT/Grafy-text07.pdf, (květen 2008)

[Ch06] CHMELAŘ, P. QM3*: Zjišťování izomrofizmu mezi grafy* [online]. 18s. 2006. Dostupný z: http://www.fit.vutbr.cz/~chmelarp/public/06grafizo.pdf, (květen 2008)

[Ic00] ICHIKAWA, S. et. al.: Evaluation of Accelerator Designs for Subgraph Isomorphism Problem. *In Proceedings of 10th Int.l Conf. on Field Programmable Logic and Applications* [online]. Springer, 2000. str. 729-738. Dostupný z: http://citeseer.ist.psu.edu/ichikawa00evaluation.html, (květen 2008)

[Inv] WWW stránky. *Graph Invariant Database* [online]. Dostupný z: http://www.public.iastate.edu/~crb002/gid/gid.html, (květen 2008)

[Ko84] KOLÁŘ, Josef.: *Grafy*. Praha: ČVUT, 1984. 231s.

[Kr01] KŘENA, Bohuslav: The Graph Isomorphism Problem, *In: Proceedings of 7th Conference Student FEI 2001*[online], Brno, 2001. Dostupný z: http://www.fit.vutbr.cz/research/view_pub.php?id=6047, (květen 2008)

[KV84] KUPER, G. M., VARDI, M. Y.: A New Approach to Database Logic. *Proc. of the 3th Symposium on Principles of Database Systems*. ACM Press [online]. 1984. str. 86-96. Dostupný z: http://portal.acm.org/citation.cfm?id=588026, (květen 2008)

[LP90] LEVENE, M., POULOVASSILIS, A.: The Hypernode Model and its Associated Query Language. In *Proc. of the 5th Jerusalem Conf. on Information technology*. IEEE Computer Society Press [online]. 1990. str. 520-530. Dostupný z: http://ieeexplore.ieee.org/iel2/307/3595/00128324.pdf, (květen 2008)

[MB95] MESSMER, B. T., BUNKE H.: *Subgraph Isomorphism in Polynomial Time*. Technical Report IAM 95-003 [online]. University of Bern, Switzerland, 1995. 33s. Dostupný z: http://citeseer.ist.psu.edu/messmer95subgraph.html, (květen 2008)

[Mc05] McKAY, Brendan D.: *nauty User's Guide* [online]. Australia, 2007. Dostupný z: http://cs.anu.edu.au/~bdm/nauty/nug.pdf, (květen 2008)

[Ni82] NILSSON, N. J.: *Principles of Artificial Intelligence*. Berlin: Springer, 1982.

[OD03] OCHODKOVÁ, E., DRÁŽDILOVÁ, P.: *Grafové algoritmy*. VŠB TU Ostrava, 2003. 57s. Dostupný z: http://www.cs.vsb.cz/ochodkova/courses/gra/gap.pdf, (květen 2008)

[Obr1] WWW stránky. Dostupný z: http://www.langeman.net/MSci620_final_paper_files/image001.jpg, (květen 2008)

[Obr2] WWW stránky. Dostupný z: http://bodmas.org/blog/maths/4-colour-theorem/, (květen 2008)

[Ull76] ULLMANN, J. R.. R.: An Algorithm for Subgraph Isomorphism. *Journal of the ACM* [online]. 1976. str. 31-42. ISSN 0004-5411. Dostupný z: http://portal.acm.org/citation.cfm?id=321925, (květen 2008)

[Ve07] VEČERKA, Arnošt.: *Grafy a grafové algoritmy* [online]. PřF UP Olomouc, 2007. 106s. Dostupný z: http://phoenix.inf.upol.cz/esf/ucebni/Grafy_a_grafove_algoritmy.pdf, (květen 2008)

[Wi08] WWW stránky. *Graph theory* [online]. 2008. Dostupný z: http://en.wikipedia.org/wiki/Graph_theory, (květen 2008)